Linked Open Data and the Semantic Web

James Smith
Welcome to DHSI 2019!

Thanks for joining the DHSI community!

In this booklet, you will find essential course materials prefaced by some useful information about getting settled initially at UVic, finding your way around, getting logged in to our network (after you’ve registered the day before our courses begin), and so on.

Given our community’s focus on things computational, it will be a surprise to no one that we might expect additional information online for some of the classes - your instructors will let you know - or that the most current version of all DHSl-related information may be found on our website at dhsi.org.

Do check in there first if you need anything that’s not in this coursepak.

To access the DHSI wifi network, simply go into your wireless settings and connect to the “DHSI” network and enter the password “dhsi2019”.

And please don’t hesitate to be in touch with us at institut@uvic.ca or via Twitter at @AlyssaA_DHSI or @DHInstitute if we can be of any help ....
Regional Map of Greater Victoria

Legend
- **Direction of Travel**
- **Route Name**
- **Transit Exchange**
- **Park & Ride Lot** (no overnight parking)
- **Major Stop**

Average Frequency
- **Regional Route**: 15–60 minute service with limited stops
- **Frequent Route**: 15 minute or better service, 7am-7pm, Mon-Fri
- **Local Route**: 20–120 minute service

Map details include regions, roads, transit stops, and symbols for different types of services.
DHSI Wi-Fi

Network name: DHSI
Passkey: dhsi2019
Psst: Some Suggested Outings

If you're here a day or two before we begin, or staying a day or two afterwards, here are a few ideas of things you might consider doing ...

Suggested Outing 1, Botanical Beach (self-organised; car needed)
A self-guided visit to the wet, wild west coast tidal shelf (and historically-significant former research site) at Botanical Beach; we recommend departing early (around 8.00 am) to catch low tide for a better view of the wonderful undersea life! Consider bringing a packed lunch to nibble-on while looking at the crashing waves when there, and then have an afternoon drink enjoying the view from the deck of the Port Renfrew Hotel.

Suggested Outing 2, Butchart Gardens (self-organised)
A shorter journey to the resplendently beautiful Butchart Gardens and, if you like, followed by (ahem) a few minutes at the nearby Church and State Winery, in the Saanich Peninsula. About an hour there by public bus from UVic, or 30 minutes by car.

Suggested Outing 3, Saltspring Island (self-organised; a full day, car/bus + ferry combo)
Why not take a day to explore and celebrate the funky, laid back, Canadian gulf island lifestyle on Saltspring Island. Ferry departs regularly from the Schwartz Bay ferry terminal, which is about one hour by bus / 30 minutes by car from UVic. You may decide to stay on forever ....

Suggested Outing 4, Paddling Victoria's Inner Harbour (self-organised)
A shorter time, seeing Victoria's beautiful city centre from the waterways that initially inspired its foundation. A great choice if the day is sunny and warm. Canoes, kayaks, and paddle boards are readily rented from Ocean River Adventures and conveniently launched from right behind the store. Very chill.

And more!
Self-organised High Tea at the Empress Hotel, scooter rentals, visit to the Royal BC Museum, darts at Christies Carriage House, a hangry breakfast at a local diner, whale watching, kayaking, brew pub sampling (at Spinnaker's, Swans, Moon Under Water, and beyond!), paddle-boarding, a tour of used bookstores, and more have also been suggested!

Sunday, 2 June 2019 [DHSI Registration + Suggested Outings]

9:00 to 4:00
Early Class Meeting: 4. [Foundations] DH For Department Chairs and Deans (David Strong Building C124, Classroom)
Further details are available from instructors in mid May to those registered in the class. Registration materials will be available in the classroom.

3:00 to 5:00
DHSI Registration (MacLaurin Building, Room A100)
After registration, many will wander to Cadboro Bay and the pub at Smuggler's Cove OR the other direction to Shelbourne Plaza and Maude Hunter's Pub OR even into the city for a nice meal.

Monday, 3 June 2019
Your hosts for the week are Alyssa Arbuckle, Ray Siemens, and Jannaya Friggstad Jensen.
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30 to 10:00</td>
<td>Welcome, Orientation, and Instructor Overview (MacLaurin A144)</td>
<td>MacLaurin A144</td>
</tr>
<tr>
<td></td>
<td>• Welcome to the Territory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Welcome to DHSI: Ray Siemens, Alyssa Arbuckle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Welcome from UVic: Jonathan Bengtson (University Librarian), Alexandra D'Arcy (Associate Dean Research, Humanities)</td>
<td></td>
</tr>
<tr>
<td>8:30 to 10:00</td>
<td>Classes in Session (click for details and locations)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 1. [Foundations] Digitisation Fundamentals and their Application (Clearihue A103, Lab)</td>
<td>Clearihue A103, Lab</td>
</tr>
<tr>
<td></td>
<td>• 2. [Foundations] Introduction to Computation for Literary Criticism (Clearihue A102, Lab)</td>
<td>Clearihue A102, Lab</td>
</tr>
<tr>
<td></td>
<td>• 3. [Foundations] Making Choices About Your Data (Digital Scholarship Commons, McPherson Library A308, Classroom)</td>
<td>Digital Scholarship Commons, McPherson Library A308, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 4. [Foundations] DH For Department Chairs and Deans (David Strong Building C124, Classroom)</td>
<td>David Strong Building C124, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 5. [Foundations] Developing a Digital Project (With Omeka) (Clearihue A031, Lab)</td>
<td>Clearihue A031, Lab</td>
</tr>
<tr>
<td></td>
<td>• 9. Out-of-the-Box Text Analysis for the Digital Humanities (Human and Social Development A160, Lab)</td>
<td>Human and Social Development A160, Lab</td>
</tr>
<tr>
<td></td>
<td>• 10. Sound and Digital Humanities (Cornett A120, Classroom)</td>
<td>Cornett A120, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 11. Critical Pedagogy and Digital Praxis in the Humanities (Clearihue D132, Classroom)</td>
<td>Clearihue D132, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 12. Digital Humanities for Japanese Culture: Resources and Methods (McPherson Library A003, Classroom)</td>
<td>McPherson Library A003, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 13. Conceptualising and Creating a Digital Edition (McPherson Library 210, Classroom)</td>
<td>McPherson Library 210, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 14. Retro Machines &amp; Media (McPherson Library 129, Classroom)</td>
<td>McPherson Library 129, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 15. Geographical Information Systems in the Digital Humanities (Clearihue A105, Lab)</td>
<td>Clearihue A105, Lab</td>
</tr>
<tr>
<td></td>
<td>• 16. Introduction to IIIF: Sharing, Consuming, and Annotating the World’s Images (Cornett A121, Classroom)</td>
<td>Cornett A121, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 17. Web APIs with Python (Human and Social Development A170, Lab)</td>
<td>Human and Social Development A170, Lab</td>
</tr>
<tr>
<td></td>
<td>• 18. Ethical Data Visualization: Taming Treacherous Data (Cornett A128, Classroom)</td>
<td>Cornett A128, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 19. Linked Open Data and the Semantic Web (Cornett A132, Classroom)</td>
<td>Cornett A132, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 20. Palpability and Wearable Computing (McPherson Library A025, Classroom)</td>
<td>McPherson Library A025, Classroom</td>
</tr>
<tr>
<td></td>
<td>• 22. The Frontend: Modern JavaScript &amp; CSS Development (Clearihue A030, Lab)</td>
<td>Clearihue A030, Lab</td>
</tr>
<tr>
<td></td>
<td>• 23. Information Security for Digital Researchers (David Strong Building C114, Classroom)</td>
<td>David Strong Building C114, Classroom</td>
</tr>
<tr>
<td>10:15 to Noon</td>
<td>Lunch break / Unconference Coordination Session (MacLaurin A144)</td>
<td>MacLaurin A144</td>
</tr>
<tr>
<td></td>
<td>(Grab a sandwich and come on down!)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Discussion topics, scheduling, and room assignments from among all DHSI rooms will be handled at this meeting.</td>
<td></td>
</tr>
<tr>
<td>1:30 to 4:00</td>
<td>Classes in Session</td>
<td></td>
</tr>
<tr>
<td>1:30 to 4:00</td>
<td>Institute Lecture: Jacqueline Wernimont (Dartmouth C): <em>Sex and Numbers: Pleasure, Reproduction, and Digital Biopower</em></td>
<td>MacLaurin A144</td>
</tr>
<tr>
<td>1:30 to 4:00</td>
<td>Chair: Anne Cong-Huyen (U Michigan)</td>
<td></td>
</tr>
<tr>
<td>4:10 to 5:00</td>
<td>Abstract: Drawing from Numbered Lives (MIT 2018), this talk will consider a long history of sex-number entanglement in Anglo-American Cultures. Drawing on historical and contemporary objects and practices, Wernimont will ask “in what ways do theories of biopower, critical gender and critical race studies, and media studies” suggest that we can understand this set of entanglements and their impacts. NB: While relevant, this talk will not include discussions of sexual trauma or violence. It will include frank discussion of sex acts and various ways of translating sexual behavior into numbers.</td>
<td></td>
</tr>
<tr>
<td>5:00 to 6:00</td>
<td>Opening Reception (University Club)</td>
<td></td>
</tr>
<tr>
<td>Tuesday, 4 June 2019</td>
<td>DHSI Conference and Colloquium Lightning Talk Session 1 (MacLaurin A144)</td>
<td>MacLaurin A144</td>
</tr>
<tr>
<td>9:00 to Noon</td>
<td>Classes in Session</td>
<td></td>
</tr>
<tr>
<td>12:15 to 1:15</td>
<td>Lunch break / Unconference</td>
<td></td>
</tr>
<tr>
<td>12:15 to 1:15</td>
<td><em>Mystery</em> Lunches</td>
<td></td>
</tr>
<tr>
<td>1:30 to 4:00</td>
<td>Classes in Session</td>
<td></td>
</tr>
<tr>
<td>4:10 to 5:00</td>
<td>Institute Lecture: Jacqueline Wernimont (Dartmouth C): <em>Sex and Numbers: Pleasure, Reproduction, and Digital Biopower</em></td>
<td>MacLaurin A144</td>
</tr>
<tr>
<td>4:10 to 5:00</td>
<td>Chair: Anne Cong-Huyen (U Michigan)</td>
<td></td>
</tr>
<tr>
<td>4:10 to 5:00</td>
<td>Abstract: Drawing from Numbered Lives (MIT 2018), this talk will consider a long history of sex-number entanglement in Anglo-American Cultures. Drawing on historical and contemporary objects and practices, Wernimont will ask “in what ways do theories of biopower, critical gender and critical race studies, and media studies” suggest that we can understand this set of entanglements and their impacts. NB: While relevant, this talk will not include discussions of sexual trauma or violence. It will include frank discussion of sex acts and various ways of translating sexual behavior into numbers.</td>
<td>MacLaurin A144</td>
</tr>
</tbody>
</table>
Wednesday, 5 June 2019

9:00 to Noon
Classes in Session

Lunch break / Unconference

"Mystery" Lunches

Presentation: An Introduction to Scholarly Publishing with Manifold (MacLaurin A144)
Lunch included for those who register here

This presentation introduces Manifold Scholarship, a Mellon-funded digital publishing platform developed by the CUNY Graduate Center, The University of Minnesota Press, and Cast Iron Coding. Manifold allows you to create beautiful, dynamic open access projects that can include text, images, video, embedded resources, and social annotation. We will provide an overview of Manifold and demonstrate how faculty, students and staff in the digital humanities can use Manifold to publish open access scholarly works, conduct and participate in peer review, and create custom edited versions of public domain course texts and OER.

1:30 to 4:00
Classes in Session

4:15 to 5:15
DHSI Conference and Colloquium Lightning Talk Session 2 (MacLaurin A144)
Chair: Kim O'Donnell (Simon Fraser U)

- Catherine Ryu (Michigan State U), "Tone Perfect: Developing a Multimodal Audio Database for Mandarin Chinese as an Open Source"
- Kenzie Burchell (U Toronto Scarborough), "Making Responsible Reporting Practices Visible: Comparing newswire coverage of humanitarian crises in Syria"
- Jessica Linzel (Brock U), "The Shopkeeper Aristocracy: Mapping Trade Networks in Colonial Niagara"
- Kirsten Painter (U Washington), "From Bogatyrs to Bread: Digitization & Online Exhibition of Rare Russian Children's Books at the U Washington"
- John Barber (Washington State U), "A Mighty Span"

6:00 to 7:00
"Half Way There!" [An Informal, Self-Organized Birds of a Feather Get-Together] (Felicitas, Student Union Building)
Bring your DHSI nametag and enjoy your first tipple on us! [A great opportunity for an interest group meet-up ....]

Thursday, 6 June 2019

9:00 to Noon
Classes in Session

Lunch break / Unconference

12:15 to 1:15
"Mystery" Lunches

[Instructor lunch meeting]

1:30 to 4:00
Classes in Session

4:15 to 5:15
DHSI Conference and Colloquium Lightning Talk Session 3 (MacLaurin A144)
Chair: Kim O'Donnell (Simon Fraser U)

- Colleen Kolba (U South Florida), "What Comics can Teach our Students about Multimodal Literacy"
- Trish Baer (ETCL; U Victoria), "Preserving Digital Legacies: Archived Websites and Digital Discoverability"
- Suchismita Dutta (U Miami), "The Importance of Archival Transcription for Genre Building"
- Jeffrey Lawler (California State U, Long Beach), "Twining our way through the Past: Video Game Authoring as History Pedagogy"
Friday, 7 June 2019 [DHSI; ADHO Pedagogy SIG Conference Opening]

9:00 to Noon  Classes in Session

12:15 to 1:15  Lunch Reception / Course E-Exhibits (MacLaurin A100)

1:30 to 1:50  Remarks, A Week in Review (MacLaurin A144)

Joint Institute Lecture (DHSI and ADHO Pedagogy SIG Conference):
Matt Gold (CUNY Graduate Center and Association for Computers and the Humanities): “Thinking Through DH: Proposals for Digital Humanities Pedagogy”
Chair: Diane Jakacki (Bucknell U)
(MacLaurin A144)

Abstract: How do we teach digital humanities, and how should DH be taught? What, indeed, should we teach when we teach DH? This talk will present a proposal for grounding digital humanities pedagogical practice in the research interests of our students and the epistemological foundations of our methods rather than through an approach grounded more central in data and methods.

Joint Reception: DHSI and ADHO Pedagogy SIG Conference (University Club)
E-Poetry Event (Chris Tanasescu)
Watch this space for details, including how to participate!

DHSI Conference and Colloquium Poster/Demo Session
- Pia Russel (U Victoria); Emily Stremel (U Victoria), “British Columbia’s Historical Textbooks Digital Library”
- Cody Hennesy (U Minnesota); Rachael Samberg (U California, Berkeley); Stacy Reardon (U California, Berkeley), “Finding the Haystack: Literacies for Accessing and Using Text as Data”
- Paula Johanson (ETCL; Independent Scholar), “Proving Seahorses and Juan de Fuca’s Travels in The Curve of Time”
- Tara Baillargeon (Marquette U); Elizabeth Wawrzyniak (Marquette U), “FellowsHub: J. R. R. Tolkien Fanzine Portal”
- Caterina Agostini (Rutgers U), “Art at the Time of Syphilis: A First-Person Medical Narrative in Benvenuto Cellini’s Vita”
- Lauren Elle DeGaine (ETCL; U Victoria), “Women at the Front: A Digital Exhibit of Victorian Frontpiece Illustrations”
- Adam Griggs (Mercer U); Kathryn Wright (Mercer U); Christian Pham (Mercer U); Gail Morton (Mercer U); Stephanie Miranda (Mercer U), “Digitizing Middle Georgia’s History of Slavery”

Saturday, 8 June 2019 [Conference, Colloquium, and Workshop Sessions]

8:00 to 9:00  Conference / Workshop Registration (MacLaurin A100)

The day’s events are included with your DHSI registration. If you’re not registered in DHSI, you’re very welcome to join us by registering here as a Conference / Colloquium / Workshop participant. We’ll have a nametag waiting for you!

Coffee, Tea, &c?
Looking for some morning coffee or tea, or a small nibble? Options and hours of operation for weekend campus catering are available here. Mystic Market usually opens around 10.00.

9:00 to 4:00  DHSI Conference and Colloquium Sessions
ADHO Pedagogy SIG Conference Sessions
Right2Left Workshop Sessions

All Day DHSI Workshop Session (click for workshop details and free registration for DHSI participants)
- 55. Introduction to Machine Learning in the Digital Humanities [8-9 June; All day, each day] (David Strong Building C124, Classroom)

9:00 to 9:10  Informal Greetings, Room Set-up (Lobby, outside Hickman 105)

Session 1
DHSI Colloquium and Conference (Hickman 105)
Digital Humanities & Literature, Chair: Kim O’Donnell (Simon Fraser U)
- Youngmin Kim (Dongguk U), “Transdiscursivity in the Convergence of Digital Humanities and World Literature”
- Caroline Winter (U Victoria), “Digitizing Adam Smith’s Literary Library”
- Kaitlyn Fralick (U Victoria); Kailey Fukushima (U Victoria); Sarah Karlson (U Victoria), “Victorian Poetry
9:10 to 10:30
ADHO Pedagogy SIG Conference (Hickman 110)
Chair: Katherine Faull (Bucknell U)
- Ashleigh McIntyre (U Newcastle), “The Language of Criticism in the Anthropocene”
- Aaron Tucker and Nada Savicevic (Ryerson U), “Write Here, Right Now: An Open Source eTextbook for the Flipped Classroom”
- Heather McAlpine (U Fraser Valley), “Digital Meters: Using Text Encoding to Teach Literature in the Undergraduate Classroom”
- Tiina H. Airaksinen (U Helsinki), “Digital Humanities in Cultural Studies: Creating a MOOC course for University Students and A-Level Students”

Right2Left Workshop (Hickman 116)

10:30 to 10:40
Break

10:40 to Noon
Session 2
DHSI Colloquium and Conference (Hickman 105)
Digital Humanities & Society, Chair: Eleanor Reed (Hastings C)
- Joel Zapata (Southern Methodist U), “Uncovering the Southern Plains’ Mexican American Civil Rights Movement”
- Brendan Mackie (U California, Berkeley), “Visualizing Long-Term Cultural Change: An Example From The Birth of Civil Society”

ADHO Pedagogy SIG Conference (Hickman 110)
Chair: Laura Estill (St Francis Xavier U)
- Jane Jackson (Chinese U of Hong Kong), “Interrogating digital spaces for intercultural meaning-making”
- Christopher Church, Katherine Hepworth (U Nevada, Reno), “We’re STEAMed! A call for balancing technical instruction and disciplinary content in the digital humanities”

Right2Left Workshop (Hickman 116)
- Edward “Eddie” Surman (Claremont Graduate U), “Qualitative Digital Text Analysis and #Right2Left Languages: A Demonstration of Atlas.ti using the Hebrew Bible”

Noon to 1:10
Lunch (We recommend Mystic Market on weekends!)

1:10 to 2:30
Session 3
DHSI Colloquium and Conference (Hickman 105)
Digital Humanities & Community, Chair: Claire Carlin (U Victoria)
- Pia Russel (U Victoria); Emily Stremel (U Victoria), “Mentorship and disability: Supporting disabled employees in digital humanities”
- Amy Lueck (Santa Clara U), “Virtually Emplacing Indigenous Memory”
- Md. Shehabul Alam (National U Bangladesh), “Integrating Library Service with Union Information and Service Center: A Joint Initiative towards Digital Bangladesh”

ADHO Pedagogy SIG Conference (Hickman 110)
Chair: Chris Tănăsescu (UC Louvain)
- Laura Estill (St Francis Xavier U), “One Assignment, Three Ways: Assessing DH Projects in a Literature Course”
- Francesca Giannetti (Rutgers U, New Brunswick), “So near while apart: Correspondence Editions as Critical Library Pedagogy and Digital Humanities Methodology”

Right2Left Workshop (Hickman 116)
- Najla Jarkas (American U Beirut) and David Joseph Wrisley (NYU Abu Dhabi), “RTL Software Localization and Digital Humanities: the Case Study of Translating Voyant Tools into Arabic”
### Sunday, 9 June 2019 [Workshop Sessions]

**DHSI Registration** *(MacLaurin Building, Room A100)*

The day’s events are included with your DHSI registration. If you’re not registered in DHSI, you’re very welcome to join us by registering here as a Conference / Colloquium / Workshop participant. We’ll have a nametag waiting for you!

**Coffee, Tea, &c?**

Looking for some morning coffee or tea, or a small nibble? Options and hours of operation for weekend campus catering are available here. Mystic Market usually opens around 10:00.

**All Day Workshop Sessions** *(click for workshop details and free registration for DHSI participants)*

<table>
<thead>
<tr>
<th>Time</th>
<th>Workshop</th>
</tr>
</thead>
</table>
| 9:00 to 4:00 | - 55. Introduction to Machine Learning in the Digital Humanities [8-9 June; All day, each day] *(David Strong Building C124, Classroom)*  
- 57. Natural Language Processing and Network Coding Apps for Text & Textual Corpus Analysis in the Humanities [9 June; All Day] *(David Strong Building C114, Classroom)* |
| 9:00 to Noon | - 59. 3D Visualization for the Humanities [9 June; AM] *(Cornett A229, Classroom)*  
- 60. It’s All Relational: AbTeC’s Indigenous Video Game Workshops as Storytelling Praxis [9 June; AM] *(Cornett A121, Classroom)*  
- 61. Spatial DH: De-Colonizing Cultural Territories Online [9 June; AM] *(Clearihue D130, Classroom)*  
| Noon to 1:00 | Lunch *(We recommend Mystic Market on weekends!)* |
| 1:00 to 4:00 | - 65. Indigenous Futurities in the Classroom and Beyond [9 June; PM] *(Cornett A121, Classroom)*  
- 66. DHSI Knits: History of Textiles and Technology [9 June; PM] *(Fine Arts 109, Classroom)*  
- 68. Linked Open Datafication for Humanities Scholars [9 June; PM] *(McPherson Library A003, Classroom)*  
- 69. Stylo - WYSIWYM Text Editor for Humanities Scholars [9 June; PM] *(McPherson Library A025, Classroom)* |

After the day, many will wander to Cadboro Bay and the pub at Smuggler’s Cove OR the other direction to Shelbourne Plaza and Maude Hunter’s Pub OR even into the city for a bite to eat.
Your hosts for the week are Ray Siemens and Jannaya Friggstad Jensen.

**7:45 to 8:15**
DHSI Last-minute Registration (MacLaurin A100)

**8:30 to 10:00**
Welcome, Orientation, and Instructor Overview (MacLaurin A144)

**10:15 to Noon**
Classes in Session (click for details and locations)
- 29. [Foundations] Understanding The Predigital Book: Technologies of Inscription (McPherson Library A003, Classroom)
- 30. [Foundations] Databases for Digital Humanists (McPherson Library 210, Classroom)
- 33. Digital Storytelling (Cornett A120, Classroom)
- 34. Text Mapping as Modelling (Clearihue D131, Classroom)
- 35. Stylometry with R: Computer-Assisted Analysis of Literary Texts (Clearihue A102, Lab)
- 36. Open Access and Open Social Scholarship (Clearihue D130, Classroom)
- 37. Digital Games as Tools for Scholarly Research, Communication and Pedagogy (Cornett A229, Classroom)
- 38. Queer Digital Humanities (David Strong Building C114, Classroom)
- 40. Introduction to Electronic Literature in DH: Research and Practice (Cornett A128, Classroom)
- 41. Surveillance and the Critical Digital Humanities (David Strong Building C108, Classroom)
- 42. Digital Humanities Pedagogy: Integration in the Curriculum (Cornett A121, Classroom)
- 43. Accessibility & Digital Environments (Priestly Law Library 265, Classroom)
- 44. Agile Project Management (Cornett A132, Classroom/Lab)
- 45. XPath for Processing XML and Managing Projects (Clearihue A105, Lab)
- 46. Endings: How to End (and Archive) your Digital Project (Priestly Law Library 192, Classroom)
- 47. Introduction to Network Analysis in the Digital Humanities (Clearihue D132, Classroom)
- 48. Text Processing - Techniques & Traditions (McPherson Library A025, Classroom)
- 49. Introduction to Humanities Data Analysis & Visualization in R (HDA) (Human and Social Development A160, Lab)
  Chair: Constante Crompton (U Ottawa)
  (MacLaurin A144)
- 51. Text Mapping as Modelling (Clearihue D131, Classroom)
  Chair: Constante Crompton (U Ottawa)
  (MacLaurin A144)

**12:15 to 1:15**
Lunch break / Unconference Coordination Session (MacLaurin A144)
(Grab a sandwich and come on down!)

**1:30 to 4:00**
Classes in Session

**4:10 to 5:00**
Institute Lecture: Angel David Nieves (San Diego State U): "3D Mapping and Forensic Traces of Testimony: Documenting Apartheid-Era Crimes Through the Digital Humanities"
Chair: Constante Crompton (U Ottawa)
(MacLaurin A144)

Abstract: In 1989 the killing of a queer, 14-year-old youth in Winnie Mandela's house named Stompie Seipei (an event that few in South Africa are willing to recall, let alone discuss, in any detail) -- is perhaps one of the most glaring examples where the queer and activist community was suppressed or erased from anti-apartheid/liberation histories. Digital humanities may actually help both reconstruct and recover a history that is still very early in the telling, despite what is commonly believed about the liberation struggle and the contributions of queer activists in the dismantling of apartheid. Perhaps it could explain why a youth such as Seipei was killed -- or at the very least, provide a more complex and messy narrative that permits one to know more about the history of queer anti-apartheid activists was suppressed. This talk outlines a methodology for "messy thinking and writing" in the digital humanities that -- through a queer and feminist intersectional framework -- permits a more complex layering of oral histories and 3D historical reconstructions.

**5:00 to 6:00**
Reception (University Club)

**Tuesday, 11 June 2019**

**9:00 to Noon**
Classes in Session

**12:15 to 1:15**
Lunch break / Unconference

**Mystery** Lunches
Wednesday, 12 June 2019

1:30 to 4:00
Classes in Session

4:15 to 5:15
DHSI Conference and Colloquium Lightning Talk Session 4 (MacLaurin A144)
Chair: Lindsey Seatter (U Victoria)
- Ashley Caranto Morford (U Toronto); Kush Patel (U Michigan); Arun Jacob (McMaster U), “#OurDHIs anti-colonial: Questions and challenges in dismantling colonial influences in digital humanities pedagogy”
- Luis Meneses (ETCL; U Victoria), “Identifying Changes in the Political Environment in Ecuador”
- Laura Horak (Carleton U), “Building the Transgender Media Portal”

6:00 to 8:00
DHSI Newcomer’s Gathering (Grad House Restaurant, Graduate Student Centre)
Come down, buy meal and a beverage, and make some new friends!

9:00 to Noon
Classes in Session

12:15 to 1:15
Lunch break / Unconference
"Mystery" Lunches
Presentation: An Introduction Jupyter Notebooks for Researchers (MacLaurin A144)
This presentation introduces Jupyter Notebooks for researchers, via a partnership between Compute Canada and the Pacific Institute for the Mathematical Sciences (PIMS) including a large number of Canadian institutions. Read more here. Presenting is James Colliander, PIMS Director and team.

1:30 to 4:00
Classes in Session

4:15 to 5:15
DHSI Conference and Colloquium Lightning Talk Session 5 (MacLaurin A144)
Chair: Lindsey Seatter (U Victoria)
- Calin Murgu (New College of Florida), “Putting local metadata to strategic use: A Dashboard for visualizing 60 years of theses metadata”
- Jason Lajoie (U Waterloo), “Queer Critical Making and the Logic of Control”
- John Barber (Washington State U), “Zambezi River Bridge”

6:00 to 7:00
"Half Way There (yet again)!" [An Informal, Self-Organized Birds of a Feather Get-Together] (Felicitas, Student Union Building)
Bring your DHSI nametag and enjoy your first tipple on us! [A great opportunity for an interest group meet-up]....

Thursday, 13 June 2019

9:00 to Noon
Classes in Session

12:15 to 1:15
Lunch break / Unconference
"Mystery" Lunches
[Instructor lunch meeting]

1:30 to 4:00
Classes in Session

4:10 to 5:00
Institute Lecture: Karina van Dalen-Oskam (Huygens Institute and U Amsterdam; Alliance of Digital Humanities Organizations): “The Riddle of Literary Quality: Some Answers”
Chair: Aaron Mauro (Penn State, Behrend C) (MacLaurin A144)
Abstract: What is literature, and can you measure it? That is the key question of the project The Riddle of Literary Quality. “The Riddle” is a research project of the Huygens Institute for the History of the Netherlands (Amsterdam) in collaboration with the Fryeke Akademy (Leeuwarden) and the Institute for Logic, Language and Computation (University of Amsterdam). The Riddle combines computational analysis of writing style with the results of a large online survey of readers, completed by almost 14,000 participants. In my talk, I will go into
some of the main results of the project.

Friday, 14 June 2019

9:00 to Noon
Classes in Session

12:15 to 1:15
Lunch Reception / Course E-Exhibits (MacLaurin A100)

1:30 to 2:00
Closing, DHSI in Review (MacLaurin A144)

Contact info:
institut@uvic.ca  P: 250-472-5401  F: 250-472-5681
Linked Open Data and the Semantic Web

DHSI 2019
Table 1 Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morning</td>
<td>How do we know what kind of data we have? What kinds of resources do we have? What kinds of questions does the data let us ask?</td>
<td>How do we create open data? What data models and file formats are available?</td>
<td>What is the RDF data model? What does RDF serialization look like?</td>
<td>How do we work with RDF? What are some RDF-friendly databases and query languages?</td>
</tr>
<tr>
<td>Afternoon</td>
<td>What is linked open data? How is this different from open data? How does this change a DH project? What is the linked open data context for your project?</td>
<td></td>
<td>What are some of the DH-related standards that use RDF?</td>
<td>How can we ask questions of the data and find answers?</td>
</tr>
<tr>
<td>Project Goals</td>
<td>Select a small number of projects that together can answer a question.</td>
<td>Which data models fit the project information and questions? What are the critical relationships your answers depend on?</td>
<td>What standards can you use to fit your data, questions, and answers to the larger world of linked open data?</td>
<td>How can you work with your data in order to answer the questions?</td>
</tr>
</tbody>
</table>

Friday morning is reserved for lab time. DHSI wraps up Friday afternoon.
Work Plan

I have designed the week so that you can discover how linked data breaks down the silo walls of project presentation. The schedule on the previous page provides the broad outline. I will tweak each day based on what you want to explore. A completed project is nice, but it’s more important at the end of the week that you have a solid understanding of how linked data and the semantic web work.

Each day, I will guide us through a few exercises concerning your project or primary interest: statement, discovery, research, and reflection. Over the course of Monday through Thursday, we will use these exercises to introduce our work to the rest of the workshop and explore shared vocabularies, connections, and questions.

Many of us are accustomed to going to the library or archive and searching through physical documents in search of exactly what we need. In the world of linked data, the world wide web is our library and archive. While I have collected many of the standards and reference documents about RDF and linked data in this workbook, the bulk of the information you need for your project is out on the web somewhere. As part of the exercises each day, you will explore the web to find the resources that you need for your project.

We should have about an hour of discussion each morning or afternoon centered on different topics as shown in the schedule. The rest of the time is ready for each of you to work through the exercises. If topics come up that are of interest to the entire workshop, we can come together for more discussion.
Exercises

The exercises are designed to run through the first four days of the workshop. Each day begins with a statement of your project in light of what you’ve learned and ends with a restatement and critique.

All of your work for these exercises should be posted to the workshop’s community. This private community allows us to share and comment on each other's work. It also serves as a community in which we can ask questions and provide feedback after the workshop is over.

Monday

1. **Introductions:** Introduce yourself to the workshop by providing a little about your background and the primary project or area of interest that you would like to focus on for the week.

2. **Projects:** Write a short description about your project and the information you are working with. Include some of the relationships that you see in the information.

3. **Discoveries:** Find a project on the web that you find interesting and that has some impact on your project. For example, it could be on the same general topic as your project, could use some techniques you’re thinking about using, or could use some of the same data you're using. Write a short post summarizing the project. Include a link to the project. Read the posts by others in the workshop and comment.

4. **Reflections:** Describe how this project makes you rethink something about your own project.

Tuesday

5. **Projects:** Rework your description from the second exercise in light of the exploration in exercises three and four. Post this to the workshop and offer feedback on others’ projects.

6. **Discussions:** Write a short description about some of the vocabularies you use in your project. Are there special lists of terms, places, people, or other things? Give some examples of these lists. If you can, provide an exhaustive enumeration of the values in one of these lists and a representative sampling for two or three other lists.
7. **Vocabularies**: Find a RDF or linked data vocabulary that contains the terms, places, people, or other things that you have in a list in your project. If you have time, find vocabularies for two or three different lists. Write a short post sharing these vocabularies with the workshop. Include links to the vocabularies.

8. **Reflections**: Describe how these vocabularies might be useful in your project. How do they reflect domain knowledge? How do they make you rethink something about your own project?

**Wednesday**

9. **Projects**: Reflect on the exploration Tuesday. Consider any changes you might want to make to your project statement, plan, or similar description. Post to the workshop and offer feedback on others’ projects.

10. **Discussions**: Write a short description about some of the connections between things in your project. Are there connections that are common across a lot of things? Do these things share many characteristics? Can they be grouped into classes based on these shared connections?

11. **Vocabularies**: Find a RDF or linked data vocabulary that describes a logical set of connections in your project. If you have time, find vocabularies for two or three different sets of connections. Write a short post sharing these vocabularies with the workshop. Include links to the vocabularies.

12. **Reflections**: Describe how these vocabularies might be useful in your project. How do they reflect domain knowledge? How do they make you rethink something about your own project?

**Thursday**

13. **Projects**: Reflect on the exploration Wednesday. Consider any changes you might want to make to your project statement, plan, or similar description. Post to the workshop and offer feedback on others’ projects.

14. **Discussions**: Write a short description about some of the questions that you can answer based on the data that you have or intend to have. How do the vocabularies and connections help you answer these questions?

15. **Discoveries**: Find another project on the web that is asking similar questions about similar data. It doesn’t need to be the same data, but if you have a project about relationships between people, then try to find another project that is
about relationships between people. It doesn't have to be an exact match, but should compare to your project in some way. Write a short post about the project and how it matches up with your project. Include links to the project and related data.

16. **Reflections:** Describe how the project in the previous exercise might make you rethink your own project. Does it ask more questions than you were planning on asking? Fewer? Does it validate what you are doing?

**Friday**

17. **Projects:** Congratulations! You’ve made it through the entire week of the workshop. In this final exercise, write a description of your project in light of all that you’ve learned.
Resources

I don't intend for this collection of resources to be exhaustive. It's a starting point in your exploration of the world of linked open data and the semantic web. We can bring more resources into the discussions based on your interests. I've ordered the sections to correspond to the schedule.

Digital Humanities Projects


The DPLA provides an API with documentation at http://dp.la/info/developers/codex/.


*Europeana* is an aggregator providing metadata about and pointers to European digital cultural objects.


*Pleiades* is a community-built gazetteer and graph of ancient places.


This archive is an example of using Open Annotation and IIIF to produce a digital facsimile edition.


*The Valley of the Shadow* is a classic example of a digital humanities project published on the web in the mid- to late-1990s.


This project provides a directory of theaters in a web 1.0 fashion. Consider how it might work if it took a linked open data approach to the problem it is solving.

This project has many facets developed over two decades showing how digital humanities has approached the challenge of presenting information on the web over that time.

**Linked Open Data**


The cloud diagram links to a large number of open data sets and shows how they are linked to each other.


Tim talks about the five stars of openness and how to approach linked data using a bag of potato chips as an example.


This resource is a good reference for how to do various data modeling tasks in linked data.

**RDF**


JSON-LD is an emerging common format for exchanging RDF-like data, especially in the context of RESTful web applications. It is the recommended serialization format for standards such as Open Annotation and Shared Canvas, used in digital facsimile editions. JSON-LD is easier to work with in JavaScript than other serializations.


This documentation describes the basics of RDF, but is most useful if you are already familiar with general data modeling concepts. Consider it a reference after attending the workshop, but feel free to look through it beforehand.

**Tools**


Jena is an open-source triple store developed under the auspices of the Apache project.


The Humanities Visualizer (HuViz) interface allows for the exploration of semantic relationships and ontologies represented using the Resource Description Framework.


Neo4j is a highly scalable, robust (fully ACID) native graph database. It doesn't support RDF, but you can work around that by pre-processing RDF-based datasets and loading them into a Neo4j instance for running queries.


These are a collection of Python libraries for working with RDF.


**Ontologies for Digital Humanities**

Shared Canvas is useful as the basis for digital facsimile editions. It comes out of work in building digital editions of medieval manuscripts. It is used by the Islandora digital edition platform to organize page images.

https://www.w3.org/community/openannotation/.

Open Annotation allows you to comment about something without having to modify the object of the commentary. Compare this to a standard such as TEI which requires inclusion of the commentary.

**DH Linked Open Data Communities**

[26] ADHO linked open data special interest group.  
http://digitalhumanities.org/lod/.

This SIG is a place for digital humanities practitioners to discuss issues around linked open data.

[27] The datahub.  
https://datahub.io/.

This website allows anyone to publish open data sets. It has had problems with spam in the past and still has some problems with stale information, but it's the largest index of open data sets.

[28] Open knowledge foundation: Open humanities.  
https://blog.okfn.org/category/open-humanities/.

This mailing list and related publications are the component of the open data movement interested in humanities data.
DH Data Guide

- Datasets
  - How to Open Up Data
    - Choose Dataset(s)
    - Apply an Open License (Legal Openness)
    - Make Data Available (Technical Openness)
    - Make Data Discoverable
  - So I've Opened Up Some Data, Now What?
    - Tell the World!
    - Getting Folks in a Room: Unconferences, Meetups, and Barcamps
    - Making Things! Hackdays, Prizes, and Prototypes
  - Contributing a Dataset
    - File Formats
    - Linked Data Ontologies
- Organizations

This section contains material from the Open Data Handbook
<http://opendatahandbook.org/> available under the CC-BY
<http://creativecommons.org/licenses/by/3.0/> license.

The notion of open data—information, public or otherwise, which anyone is free to access and re-use for any purpose—has been around for some years. In 2009, open data started to become visible in the mainstream, with various governments (such as the USA, UK, Canada and New Zealand) announcing new initiatives towards opening up their public information. Open data is becoming the third aspect of scholarly publishing alongside open source software and open access journal articles.

This book explains the basic concepts of ‘open data’, especially in relation to digital humanities. It covers how open data creates value and can have a positive impact in many different areas. In addition to exploring the background, the handbook also provides concrete information on how to produce open data.

The content of this book is composed of original material and material drawn from a variety of sources licensed under creative commons licenses. We try to provide attribution where we can. In many cases, attribution will consist of a brief paragraph at the top of a section linking to the sources we used in composing the content for that section.

Target Audience

This book has a broad audience:
for those who have never heard of open data before and those who consider themselves seasoned 'data professionals' for digital humanists for data geeks and those who have never heard of an API.

Most of the information currently provided is focused on data held by digital textual scholarship projects. However, the authors intend to broaden this as time permits. You are welcome to participate to help us with that effort.

This book is intended for those with little or no knowledge of the topic. If you do find a piece of jargon or terminology with which you aren't familiar, please see the detailed Glossary and FAQs (frequently asked questions) which are found at the end of the book.

Credits

Contributing authors

- James Smith

Existing sources directly used

- Open Data Handbook <http://opendatahandbook.org/>
Datasets

This section contains material from the Open Data Handbook <http://opendatahandbook.org/> available under the CC-BY <http://creativecommons.org/licenses/by/3.0/> license.

Open data is a tremendous resource that is as yet largely untapped. Many individuals and organisations collect or generate a broad range of different types of data in the course of their research. Scholarly data is particular significant in this respect because of the quality of the data. The focused experience of the scholar allows them to make informed decisions on which data to include or exclude, which naming authorities to use, and what normalization should be applied to the data. The resulting data reflects these scholarly activities.

There are many areas where we can expect open data to be of value, and where examples of how it has been used already exist. At the same time it is impossible to predict precisely how and where value will be created in the future. The nature of innovation is that developments often come from unlikely places.

It is already possible to point to a number of areas where open data is creating value. Some of these areas include:

- Participation
- Self-empowerment
- Improved or new products and services
- Innovation
- Improved efficiency, effectiveness, and impact of research projects
- New knowledge from combined data sources and patterns in large data volumes

Economically, open data is of great importance as well. Several studies have estimated the economic value of open data at several tens of billions of Euros annually in the EU alone. New products and companies are re-using open data. Google Translate uses the enormous volume of EU documents that appear in all European languages to train the translation algorithms, thus improving its quality of service.

Open data is also of value for government itself. For example, it is making government more effective. The Dutch department for cultural heritage is actively releasing their data and collaborating with amateur historical societies and groups such as the Wikimedia Foundation in order to execute their own tasks more effectively. This not only results in improvements to the quality of their data, but will also ultimately make the department smaller.

While there are numerous instances of the ways in which open data is already creating both social and economic value, we don’t yet know what new things will become possible. New combinations of data can create new knowledge and insights, which can lead to whole new fields of application. We have seen this in the past, for example when Dr.
Snow discovered the relationship between drinking water pollution and cholera in London in the 19th century, by combining data about cholera deaths with the location of water wells. This led to the building of London’s sewage systems, and hugely improved the general health of the population. We are likely to see such developments happening again as unexpected insights flow from the combination of different open data sets.

This untapped potential can be unleashed if we turn scholarly data into open data. This will only happen, however, if it is really open, i.e. if there are no restrictions (legal, financial or technological) to its re-use by others. Every restriction will exclude people from re-using the scholarly data, and make it harder to find valuable ways of doing that. For the potential to be realized, scholarly data needs to be open data.

**What is Open Data?**

For our purposes, open data is as defined by the [Open Definition](http://opendefinition.org/):

> Open data is data that can be freely used, reused and redistributed by anyone—subject only, at most, to the requirement to attribute and sharealike.

The full Open Definition gives precise details as to what this means. To summarize the most important:

- **Availability and Access**: the data must be available as a whole and at no more than a reasonable reproduction cost, preferably by downloading over the internet. The data must also be available in a convenient and modifiable form.

- **Reuse and Redistribution**: the data must be provided under terms that permit reuse and redistribution including the intermixing with other datasets.

- **Universal Participation**: everyone must be able to use, reuse and redistribute—there should be no discrimination against fields of endeavour or against persons or groups. For example, ‘non-commercial’ restrictions that would prevent ‘commercial’ use, or restrictions of use for certain purposes (e.g. only in education), are not allowed.

If you’re wondering why it is so important to be clear about what open means and why this definition is used, there’s a simple answer: **interoperability**.

Interoperability denotes the ability of diverse systems and organizations to work together (inter-operate). In this case, it is the ability to interoperate—or intermix—different datasets.

Interoperability is important because it allows for different components to work together. This ability to componentize and to ‘plug together’ components is essential to building large, complex systems. Without interoperability this becomes near impossible
—as evidenced in the most famous myth of the Tower of Babel where the (in)ability to communicate (to interoperate) resulted in the complete breakdown of the tower-building effort.

We face a similar situation with regard to data. The core of a “commons” of data (or code) is that one piece of “open” material contained therein can be freely intermixed with other “open” material. This interoperability is absolutely key to realizing the main practical benefits of “openness”: the dramatically enhanced ability to combine different datasets together and thereby to develop more and better products and services.

Providing a clear definition of openness ensures that when you get two open datasets from two different sources, you will be able to combine them together, and it ensures that we avoid our own ‘tower of babel’: lots of datasets but little or no ability to combine them together into the larger systems where the real value lies.

How to Open Up Data

This section contains material from the Open Data Handbook
<http://opendatahandbook.org/> available under the CC-BY
<http://creativecommons.org/licenses/by/3.0/> license.

This section gives concrete, detailed advice on how data holders can open up data. We’ll go through the basics, but also cover the pitfalls. Lastly, we will discuss the more subtle issues that can arise.

There are three key rules we recommend following when opening up data:
• **Keep it simple.** Start out small, simple and fast. There is no requirement that every dataset must be made open right now. Starting out by opening up just one dataset, or even one part of a large dataset, is fine—of course, the more datasets you can open up the better.

Remember this is about innovation. Moving as rapidly as possible is good because it means you can build momentum and learn from experience—innovation is as much about failure as success and not every dataset will be useful.

• **Engage early and engage often.** Engage with actual and potential users and reusers of the data as early and as often as you can, be they citizens, scholars, or developers. This will ensure that the next iteration of your service is as relevant as it can be.

It is essential to bear in mind that much of the data will not reach ultimate users directly, but rather via ‘info-mediaries’. These are the people who take the data and transform or remix it to be presented. For example, most of us don’t want or need a large database of GPS coordinates. We would much prefer a map. Thus, engage with info-mediaries first. They will reuse and repurpose the material.

• **Address common fears and misunderstandings.** This is especially important if you are working with or within large institutions such as universities. When opening up data you will encounter plenty of questions and fears. It is important to (a) identify the most important ones and (b) address them at as early a stage as possible.

There are four main steps in making data open, each of which will be covered in detail below. These are in very approximate order—many of the steps can be done simultaneously.

1. **Choose your dataset(s).** Choose the dataset(s) you plan to make open. Keep in mind that you can (and may need to) return to this step if you encounter problems at a later stage.

2. **Apply an open license.**
   1. Determine what intellectual property right exist in the data.
   2. Apply a suitable ‘open’ license that licenses all of these rights and supports the definition of openness discussed in the section above introducing this chapter on ‘Datasets’.
   3. N.B.: If you can’t do this, go back to step 1 and try a different dataset.

3. **Make the data available** in bulk and in a useful format. You may also wish to consider alternative ways of making it available such as via an API.

4. **Make it discoverable.** Post on the web and perhaps organize a central catalogue to list your open datasets.

Choose Dataset(s)
This section contains material from the Open Data Handbook
<http://opendatahandbook.org/> available under the CC-BY
<http://creativecommons.org/licenses/by/3.0/> license.

This section gives concrete, detailed advice on how data holders can open up data. We’ll go through the basics, but also cover the pitfalls. Lastly, we will discuss the more subtle issues that can arise.

There are three key rules we recommend following when opening up data:

- **Keep it simple.** Start out small, simple and fast. There is no requirement that every dataset must be made open right now. Starting out by opening up just one dataset, or even one part of a large dataset, is fine—of course, the more datasets you can open up the better.

  Remember this is about innovation. Moving as rapidly as possible is good because it means you can build momentum and learn from experience—innovation is as much about failure as success and not every dataset will be useful.

- **Engage early and engage often.** Engage with actual and potential users and reusers of the data as early and as often as you can, be they citizens, scholars, or developers. This will ensure that the next iteration of your service is as relevant as it can be.

  It is essential to bear in mind that much of the data will not reach ultimate users directly, but rather via ‘info-mediaries’. These are the people who take the data and transform or remix it to be presented. For example, most of us don’t want or need a large database of GPS coordinates. We would much prefer a map. Thus, engage with info-mediaries first. They will reuse and repurpose the material.

- **Address common fears and misunderstandings.** This is especially important if you are working with or within large institutions such as universities. When opening up data you will encounter plenty of questions and fears. It is important to (a) identify the most important ones and (b) address them at as early a stage as possible.

There are four main steps in making data open, each of which will be covered in detail below. These are in very approximate order—many of the steps can be done simultaneously.

1. **Choose your dataset(s).** Choose the dataset(s) you plan to make open. Keep in mind that you can (and may need to) return to this step if you encounter problems at a later stage.

2. **Apply an open license.**
   1. Determine what intellectual property right exist in the data.
   2. Apply a suitable ‘open’ license that licenses all of these rights and supports the definition of openness discussed in the section above introducing this chapter on ‘Datasets’.
3. N.B.: If you can’t do this, go back to step 1 and try a different dataset.

3. **Make the data available** in bulk and in a useful format. You may also wish to consider alternative ways of making it available such as via an API.

4. **Make it discoverable.** Post on the web and perhaps organize a central catalogue to list your open datasets.

### Apply an Open License (Legal Openness)

This section contains material from the [Open Data Handbook](http://opendatahandbook.org/) available under the [CC-BY](http://creativecommons.org/licenses/by/3.0/) license.

In most jurisdictions there are intellectual property rights in data that prevent third-parties from using, reusing and redistributing data without explicit permission. Even in places where the existence of rights is uncertain, it is important to apply a license simply for the sake of clarity. Thus, **if you are planning to make your data available you should put a license on it**—and if you want your data to be open <http://opendefinition.org/> this is even more important.

What licenses can you use? We recommend that for ‘open’ data you use one of the licenses conformant with the [Open Definition](http://opendefinition.org/) and marked as suitable for data. This list (along with instructions for usage) can be found at:

- [http://opendefinition.org/licenses/](http://opendefinition.org/licenses/)

A short 1-page instruction guide to applying an open data license can be found on the Open Data Commons site:

- [http://opendatacommons.org/guide/](http://opendatacommons.org/guide/)

### Make Data Available (Technical Openness)

This section contains material from the [Open Data Handbook](http://opendatahandbook.org/) available under the [CC-BY](http://creativecommons.org/licenses/by/3.0/) license.

Open data needs to be technically open as well as legally open. Specifically, the data needs to be available in bulk in a machine-readable format.
• **Available** Data should be priced at no more than a reasonable cost of reproduction, preferably as a free download from the Internet. This pricing model is achieved because your project should not undertake any cost when it provides data for use.

• **In bulk** The data should be available as a complete set. If you have a register which is collected under statute, the entire register should be available for download. A web API or similar service may also be very useful, but they are not substitutes for bulk access.

• **In an open, machine-readable format** Re-use of data held by the public sector should not be subject to patent restrictions. More importantly, making sure that you are providing machine-readable formats allows for greatest re-use. To illustrate this, consider statistics published as PDF documents, often used for high quality printing. While these statistics can be read by humans, they are very hard for a computer to use. This greatly limits the ability for others to reuse that data.

Here are a few policies that can help you achieve the greatest impact with your data:

• Keep it simple,
• Move fast, and
• Be pragmatic.

In particular, it is better to give out raw data now than perfect data in six months’ time.

There are many different ways to make data available to others. The most natural in the Internet age is online publication. There are many variations to this model. At its most basic, projects make their data available via their websites and a central catalogue directs visitors to the appropriate source. However, there are alternatives.

When connectivity is limited or the size of the data extremely large, distribution via other formats can be warranted. This section will also discuss alternatives, which can act to keep prices very low.

**Online methods**

Via your existing website

The system which will be most familiar to your web content team is to provide files for download from webpages. Just as you currently provide access to discussion documents, data files are perfectly happy to be made available this way.

One difficulty with this approach is that it is very difficult for an outsider to discover where to find updated information. This option places some burden on the people creating tools with your data.

Via 3rd party sites

Many repositories have become hubs of data in particular fields. For example, pachube.com is designed to connect people with sensors to those who wish to access data from them. Sites like Infochimps.com and Talis.com allow public sector agencies to store massive quantities of data for free.
Third party sites can be very useful. The main reason for this is that they have already pooled together a community of interested people and other sets of data. When your data is part of these platforms, a type of positive compound interest is created.

Wholesale data platforms such as institutional digital libraries already provide the infrastructure which can support the demand. They often provide analytics and usage information. For most scholarly projects, they are generally free.

These platforms can have two costs. The first is independence. Your project needs to be able to yield control to others. This is often politically, legally, or operationally difficult. The second cost may be openness. Ensure that your data platform is agnostic about who can access it. Software developers and scholars use many operating systems, from smartphones to supercomputers. They should all be able to access the data.

Via FTP servers

A less fashionable method for providing access to files is via the File Transfer Protocol (FTP). This may be suitable if your audience is technical, such as software developers and scholars. The FTP system works in place of HTTP, but is specifically designed to support file transfers.

FTP has fallen out of favour. Rather than providing a website, looking through an FTP server is much like looking through folders on a computer.

As torrents

BitTorrent is a system which has become familiar to policy makers because of its association with copyright infringement. BitTorrent uses files called torrents, which work by splitting the cost of distributing files between all of the people accessing those files. Instead of servers becoming over loaded, the supply increases with the demand increases. This is the reason that this system is so successful for sharing movies. It is a wonderfully efficient way to distribute very large volumes of data.

As an API

Data can be published via an Application Programming Interface (API). These interfaces have become very popular. They allow programmers to select specific portions of the data, rather than providing all of the data in bulk as a large file. APIs are typically connected to a database which is being updated in real-time. This means that making information available via an API can ensure that it is up to date.

Publishing raw data in bulk should be the primary concern of all open data initiatives. There are a number of costs to providing an API:

1. The price. They require much more development and maintenance than providing files.
2. The expectations. In order to foster a community of users behind the system, it is important to provide certainty. When things go wrong, you will be expected to incur the costs of fixing them.

Access to bulk data ensures that:
1. there is no dependency on the original provider of the data, meaning that if a re-
structure or budget cycle changes the situation, the data are still available.
2. anyone else can obtain a copy and redistribute it. This reduces the cost of distribution
away from the source agency and means that there is no single point of failure.
3. others can develop their own services using the data, because they have certainty
that the data will not be taken away from them.

Providing data in bulk allows others to use the data beyond its original purposes. For example, it allows it to be converted into a new format, linked with other resources, or versioned and archived in multiple places. While the latest version of the data may be made available via an API, raw data should be made available in bulk at regular intervals.

For example, the Eurostat statistical service has a bulk download facility offering over 4000 data files. It is updated twice a day, offers data in Tab- separated values (TSV) format, and includes documentation about the download facility as well as about the data files.

Another example is the District of Columbia OCTO’s Data Catalogue, which allows data to be downloaded in CSV and XLS format in addition to live feeds of the data.

Make Data Discoverable

This section contains material from the Open Data Handbook
<http://opendatahandbook.org/> available under the CC-BY
<http://creativecommons.org/licenses/by/3.0/> license.

Open data is nothing without users. You need to be able to make sure that people can find the source material. This section will cover different approaches.

Existing tools

There are a number of tools which are live on the web that are specifically designed to make data more discoverable.

One of the most prominent is the DataHub <http://datahub.io/> and is a catalog and data store for datasets from around the world. The site makes it easy for individuals and organizations to publish material and for data users to find material they need.

DHData.Org is an example of the dozens of specialist catalogs for different sectors and places. Many scientific communities have created a catalog system for their fields, as data are often required for publication. DHData.Org is the catalog system for digital humanities.

For institutions

In government, the emerged, orthodox practice is for a lead agency to create a catalog for the government’s data. A similar institutional pattern would be for a component of the institution such as the library to act as the lead agency and create a catalog for data gen-
erated by the institution and the projects of the institution’s faculty. When establishing a catalog, try to create some structure which allows many departments and projects to easily keep their own information current.

Resist the urge to build the software to support the catalogue from scratch. There are free and open source software solutions (such as CKAN <http://ckan.org/>) which have been adopted by many governments already. As such, investing in another platform may not be needed. If you build the catalogue using CKAN, it is easy for DData.Org to harvest metadata about the catalogued datasets.

There are a few things that most open data catalogues miss. Your program could consider the following:

- Providing an avenue to allow related private and community sectors to add their data. It may be worthwhile to think of the catalogue as the intellectual community’s catalogue, rather than the institution’s.
- Facilitating improvement of the data by allowing derivatives of datasets to be catalogued. For example, someone may geocode addresses and may wish to share those results with everybody. If you only allow single versions of datasets, these improvements remain hidden.
- Be tolerant of your data appearing elsewhere. That is, content is likely to be duplicated to communities of interest. If you have text transcription data available, then your data may appear in a catalog for literary scholars.
- Ensure that access is equitable. Try to avoid creating a privileged level of access for officials or tenured researchers as this will undermine community participation and engagement.

So I’ve Opened Up Some Data, Now What?

This section contains material from the Open Data Handbook <http://opendatahandbook.org/> available under the CC-BY <http://creativecommons.org/licenses/by/3.0/> license.

We’ve looked at how to make scholarly information legally and technically reusable. The next step is to encourage others to make use of that data.

This section looks at additional things which can be done to promote data re-use.

Tell the World!

This section contains material from the Open Data Handbook <http://opendatahandbook.org/> available under the CC-BY <http://creativecommons.org/licenses/by/3.0/> license.

First and foremost, make sure that you promote the fact that you’ve embarked on a campaign to promote open data in your area of responsibility.
If you open up a bunch of datasets, it’s definitely worth spending a bit of time to make sure that people know (or at least can find out) that you’ve done so.

In addition to things like press releases, announcements on your website, and so on, you may consider:

- Contacting prominent organisations or individuals who work/are interested in this area
- Contacting relevant mailing lists or social networking groups
- Directly contacting prospective users who you know may be interested in this data
- Create an index page on this site (/dhdata/datasets/3-contributing-a-dataset/)

Understanding your audience

Like all public communication, engaging with the data community needs to be targeted. Like all stakeholder groups, the right message can be wasted if it is directed to the wrong area.

Digital communities tend to be very willing to share new information, yet they very rapidly consume it. Write as if your messages will be skimmed over, rather than critically examined in-depth.

Members of the tech community are less likely than the general public to use MS Windows. This means that you should not save documents in MS Office formats which can be read offline. There are two reasons for this:

- The first is that those documents will be less accessible. Rather than the document you see on your screen, readers may see an imperfect copy from an alternative.
- Secondly, your project sends an implicit message that you are unwilling to take a step towards developers. Instead, you show that you are expecting the technology community to come to you.

Post your material on third-party sites

Many blogs have created a large readership in specialised topic areas. It may be worthwhile adding an article about your initiative on their site. These can be mutually beneficial. You receive more interest and they receive a free blog post in their topic area.

Making your communications more social-media friendly

It’s unrealistic to expect that all scholars should spend long periods of time engaging with social media, though many in the digital humanities community do engage via Twitter and other social media platforms. However, there are several things that you can do to make sure that your content can be easily shared between technical users. Some tips:
• **Provide unique pages for each piece of content.** When a message is shared with others, the recipient of the referral will be looking for the relevant content quickly.

• **Avoid making people download your press releases.** Press releases are fine. They are concise messages about a particular point. However, if you require people to download the content and for it to open outside of a web browser, then fewer people will read it. Search engines are less likely to index the content. People are less likely to click to download.

• **Consider using an Open license**<br>  <http://opendefinition.org/licenses/#content> for your content. Apart from providing certainty to people who wish to share your content that this is permissible, you send a message that your project understands openness. This is bound to leave an impression far more significant to proponents of open data than any specific sentence in your press release.

**Social media**

It’s inefficient for cash-strapped projects to spend hours on social media sites. The most significant way that your voice can be heard through these fora is by making sure that blog posts are easily shareable. That means, before reading the next section, make sure that you have read the last. With that in mind, here are a few suggestions:

• **Discussion fora.** Twitter has emerged as the platform of choice for disseminating information rapidly. Anything tagged with #opendata or #digitalhumanities will be immediately seen by hundreds or thousands. LinkedIn has a large selection of groups which are targeted towards open data. While Facebook is excellent for a general audience, it has not received a great deal of attention in the open data or digital humanities communities.

• **Link aggregators.** Submit your content to the equivalent of newswires for digital humanists or geeks. Register your project blog or twitter handle with Digital Humanities Now <http://digitalhumanitiesnow.org/> to reach the digital humanities community. Reddit and Hacker News are the two biggest in this arena at the moment. To a lesser extent, Slashdot and Digg are also useful tools in this area. These sites have a tendency to drive significant traffic to interesting material. They are also heavily focused on topic areas.

**Getting Folks in a Room: Unconferences, Meetups, and Barcamps**

*This section contains material from the Open Data Handbook*<br>  <http://opendatahandbook.org/> available under the CC-BY <http://creativecommons.org/licenses/by/3.0/> license.

Face-to-face events can be a very effective way to encourage others to use your data. Reasons that you may consider putting on an event include:
• Finding out more about prospective re-users
• Finding out more about demand for different datasets
• Finding out more about how people want to reuse your data
• Enabling prospective reusers to find out more about what data you have
• Enabling prospective users to meet each other (e.g. so they can collaborate)
• Exposing your data to a wider audience (e.g. from blog posts or media coverage
  that the event may help to generate)

There are also lots of different ways of running events, and different types of events, de-
pending on what aim you want to achieve. As well as more traditional conference mod-
els, which will include things like prepared formal talks, presentations and demon-
strations, there are also various kinds of participant driven events, where those who turn
up may:

• Guide or define the agenda for the event
• Introduce themselves, talk about what they’re interested in and what they’re work-
ing on, on an ad hoc basis
• Give impromptu micro-short presentations on something they are working on
• Lead sessions on something they are interested in

There is plenty of documentation online about how to run these kinds of events, which
you can find by searching for things like: ‘unconference’, ‘barcamp’, ‘meetup’,
’speedgeek’, ‘lightning talk’, and so on. You may also find it worthwhile to contact people
who have run these kinds of events in other countries, who will most likely be keen to
help you out and to advise you on your event. It may be valuable to partner with another
organisation (e.g. a civic society organisation, a news organisation or an educational in-
stitution) to broaden your base participants and to increase your exposure.

Digital humanities venues

The digital humanities community has a number of opportunities for sharing open data.

• CFPs <http://digitalhumanitiesnow.org/tag/cfp/>
• Conferences <http://digitalhumanitiesnow.org/tag/conferences/>
• THATCamp <http://thatcamp.org/>

Making Things! Hackdays, Prizes, and Prototypes

This section contains material from the Open Data Handbook
<http://opendatahandbook.org/> available under the CC-BY
<http://creativecommons.org/licenses/by/3.0/> license.

The structure of these competitions is that a number of datasets are released and pro-
grammers then have a short time-frame—running from as little as 48 hours to a few
weeks—to develop applications using the data. A prize is then awarded to the best appli-
cation. Competitions have been held in a number of countries including the UK, the US,
Norway, Australia, Spain, Denmark and Finland.

Examples of competitions
Show us a better way was the first such competition in the world. It was initiated by the UK Government’s “The Power of Information Taskforce” headed by Cabinet Office Minister Tom Watson in March 2008. This competition asked “What would you create with public information?” and was open to programmers from around the world, with a tempting £80,000 prize for the five best applications.

Apps for Democracy, one of the first competitions in the United States, was launched in October 2008 by Vivek Kundra, at the time Chief Technology Officer (CTO) of the District of Columbia (DC) Government. Kundra had developed the groundbreaking DC data catalogue, http://data.octo.dc.gov/ <http://data.octo.dc.gov/> , which included datasets such as real-time crime feeds, school test scores, and poverty indicators. It was at the time the most comprehensive local data catalogue in the world. The challenge was to make it useful for citizens, visitors, businesses and government agencies of Washington, DC.

The creative solution was to create the Apps for Democracy contest. The strategy was to ask people to build applications using the data from the freshly launched data catalogue. It included an online submission for applications, many small prizes rather than a few large ones, and several different categories as well as a “People’s Choice” prize. The competition was open for 30 days and cost the DC government $50,000. In return, a total of 47 iPhone, Facebook and web applications were developed with an estimated value in excess of $2,600,000 for the local economy.

The Abre Datos (Open Data) Challenge 2010. Held in Spain in April, 2010, this contest invited developers to create open source applications making use of public data in just 48 hours. The competition had 29 teams of participants who developed applications that included a mobile phone programme for accessing traffic information in the Basque Country, and for accessing data on buses and bus stops in Madrid, which won the first and second prizes of €3,000 and €2,000 respectively.

Nettskap 2.0. In April, 2010, the Norwegian Ministry for Government Administration held “Nettskap 2.0”. Norwegian developers—companies, public agencies or individuals—were challenged to come up with web-based project ideas in the areas of service development, efficient work processes, and increased democratic participation. The use of government data was explicitly encouraged. Though the application deadline was just a month later, on May 9, the Minister Rigmor Aasrud said the response was “overwhelming”. In total, 137 applications were received, no less than 90 of which built on the reuse of government data. A total amount of NOK 2.5 million was distributed among the 17 winners; while the total amount applied for by the 137 applications was NOK 28.4 million.

Mashup Australia. The Australian Government 2.0 Taskforce invited citizens to show why open access to Australian government information would be positive for the country’s economy and social development. The contest ran from October 7th to November 13th, 2009. The Taskforce released some datasets under an open licence and in a range
of reusable formats. The 82 applications that were entered into the contest are further evidence of the new and innovative applications which can result from releasing government data on open terms.

**Contributing a Dataset**

Contributing information about a dataset is as simple as forking [this GitHub repository](https://github.com/dhdata/dhdata-site), writing a short page about the dataset, and creating a pull request [here](https://help.github.com/articles/using-pull-requests). Once the request is accepted, the dataset will be published on the site.

The easiest way to write a dataset page is to use the [editor](http://prose.io) based on Prose.io. Once you are authenticated, you will see a list of available content categories. If you have not yet forked the repository, then the editor will show you a short list of instructions with links to the appropriate GitHub pages.

To create an index page for a dataset, follow these steps.

**Step 1: Open the datasets folder**

All dataset pages are in the datasets folder. Selecting the folder will list all of the datasets.

```
- dhdata-site

  - categories
  - datasets
  - groups
  - organizations
  - recipes

Step 2: Create a new document```
Selecting the **New File** button at the top of the page will open up an edit window with the dataset template loaded.

Step 3: **Describe your dataset**

The first section of the document is where you should describe your dataset. This section should not have any section headings. If you feel the need to group together parts of your dataset into subsections, then you should break your dataset into multiple sets with one page per set.

Step 4: **List your resources**

The second section is where you list each of the resources that make up your dataset. Each resource should follow the template:

- a level three heading naming the resource,
- a short description of the resource, and
- a table of metadata about the resource.

The resource name should be unique within the dataset. DHData.Org uses this name to build out URLs for resource-specific pages such as any resource preview.

The table should map the metadata keys to the appropriate values. DHData.Org uses only the first two columns for each row, so you are free to add comments in a third column. However, the table is not shown on the site as-is, so any additional columns will only be seen in the repository.

The following keys are used by DHData.Org. Additional metadata keys are shown on the site but are not used.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
type Indicates the broad format class. This should be one of: Collection, Dataset, Interactive Resource, Moving Picture, Music, Service, Still Image, or Text.

url Indicates the web address from which the resource may be retrieved. Fetching this URL should result in a file having the same MIME type as indicated in this metadata. Likewise, the file should contain data in the indicated format.

File Formats

This section contains material from the Open Data Handbook <http://opendatahandbook.org/> and DCMI Metadata Terms <http://dublincore.org/documents/dcmi-terms/> available under the CC-BY <http://creativecommons.org/licenses/by/3.0/> license.

Because the purpose of DH Data is to share open data and computable data, we accept a limited list of file formats and types. If your data is not in one of the formats listed on this page, it is not sufficiently open to be useful to the larger digital humanities community without conversion. This is not an indictment of the content, but of the particular file format used to repesent that content.

These resource data formats use the DCMI Type Vocabulary for the primary classification followed by the file’s serialization format. For each format, we’ve linked to any standards documents and indicated the range of openness stars a typical resource can earn.

Collection

Collections are resources aggregating other resources. For example, an RSS feed aggregates blog posts.

When defining resource metadata in a dataset index page, the dominant type of the resource within the collection should be combined with the word “Collection” to indicate the resource type. For example, a TAR file containing TEI files would have a type of “Text Collection” because TEI files are Text resources.

<table>
<thead>
<tr>
<th>Format</th>
<th>Specification</th>
<th>Openness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>ORE</td>
<td>Specification <a href="http://www.openarchives.org/ore/1.0/rdfxml.html">http://www.openarchives.org/ore/1.0/rdfxml.html</a></td>
<td>★★★(★★)</td>
</tr>
<tr>
<td>RSS</td>
<td>Specifications <a href="http://www.rss-specifications.com/">http://www.rss-specifications.com/</a></td>
<td>*(★★★★)</td>
</tr>
</tbody>
</table>
An archive is a collection of resources that are compiled into a single resource using a structured format such as TAR or ZIP. When creating a dataset, the format metadata key can be set to the format of the individual resources concatenated with “Archive.” For example, a TAR file containing TEI files can have a format of “TEI Archive.”

Dataset

Datasets are serializations of tabular or graph data such as tables, lists, or databases. Datasets are useful for direct computation.

<table>
<thead>
<tr>
<th>Format</th>
<th>Specification</th>
<th>Openness</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON-LD</td>
<td>Specifications <a href="http://json-id.org/spec/latest/">http://json-id.org/spec/latest/</a></td>
<td>★★★★★(*)</td>
</tr>
<tr>
<td>KML</td>
<td>Specification <a href="http://www.opengeospatial.org/standards/kml/">http://www.opengeospatial.org/standards/kml/</a></td>
<td>★★★(**)</td>
</tr>
<tr>
<td>OWL/XML</td>
<td></td>
<td>★★★★★</td>
</tr>
<tr>
<td>RDF/JSON</td>
<td></td>
<td>★★★(*★)</td>
</tr>
<tr>
<td>RDF/N3</td>
<td></td>
<td>★★★(*★)</td>
</tr>
<tr>
<td>RDF/N-Triples</td>
<td></td>
<td>★★★(*★)</td>
</tr>
<tr>
<td>RDF/Turtle</td>
<td></td>
<td>★★★(*★)</td>
</tr>
<tr>
<td>RDF/XML</td>
<td></td>
<td>★★★(*★)</td>
</tr>
<tr>
<td>TSV</td>
<td>Specification <a href="http://www.iana.org/assignments/mediatypes/text/tab-separated-values">http://www.iana.org/assignments/mediatypes/text/tab-separated-values</a></td>
<td>★★★</td>
</tr>
</tbody>
</table>

Comma and Tab Separated Files

CSV files can be a very useful format because they are compact and thus suitable for transferring large sets of data with the same structure. However, the format is so spartan that data are often useless without documentation. It can be almost impossible to guess the significance of the different columns without some knowledge about the context of the data. It is therefore particularly important for the comma-separated formats that documentation of the individual fields be accurate.

Furthermore, it is essential that the structure of the file be respected, as a single omission of a field may disturb how an application interprets the remaining data in the file without any real opportunity to fix it because the application cannot determine how the
remaining data should be interpreted.

**JSON**

JSON is a simple file format that is very easy for any programming language to read. Its simplicity means that it is generally easier for computers to process than others, such as XML.

**JSON-LD**

JSON-LD is a form of JSON with contextual information that helps an application know how to interpret the data. The JSON-LD data model is comparable to RDF, but it is not a simple JSON serialization of RDF.

**RDF**

A W3C-recommended format called RDF (the Resource Description Framework) makes it possible to represent data in a way that makes it easier to combine data from multiple sources. RDF data can be stored in XML and JSON, among other serializations. RDF encourages the use of URLs as identifiers, which provides a convenient way to directly interconnect existing open data initiatives on the Web. RDF is still not widespread, but it has been a trend among Open Government initiatives, including the British and Spanish Government Linked Open Data projects. Recent developments in digital facsimile projects use RDF-based data models such as Open Annotation and Shared Canvas. The inventor of the Web, Tim Berners-Lee, has recently proposed a five-star <http://lab.linkeddata.deri.ie/2010/star-scheme-example/> scheme that includes linked RDF data as a goal to be sought for open data initiatives.

**XML**

XML is a widely used format for data exchange because it provides good opportunities for building structure in data. Developers can mix parts of the documentation in with the data without interfering with the reading of either. XML is used as a serialization format for RDF as well as the native document format for (X)HTML, MEI, and TEI.

**Image**

A visual representation other than text. Image resources should not be “lossy” unless their particular compression demonstrates an issue with the compression or decompression algorithm. They should be useful for computation. For example, an exemplar image demonstrating some particular difficulty for a particular algorithm would be a useful Image resource.

Any dataset composed of Images must explain in the description why the dataset is useful.

<table>
<thead>
<tr>
<th>Format</th>
<th>Specification</th>
<th>Openness</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIF</td>
<td>Specification <a href="http://www.w3.org/Graphics/GIF/spec-gif89a.txt">http://www.w3.org/Graphics/GIF/spec-gif89a.txt</a></td>
<td>✫</td>
</tr>
</tbody>
</table>
Interactive Resource

- Openness: ★

A resource that requires a user to interact with it for it to be understood or useful. Examples include search pages that are not designed as a simple visual representation of a REST API. We assume that Interactive Resources are HTML pages.

Moving Picture

- Openness: ★

As with Images, Moving Pictures are not considered open data unless they serve as exemplars for particular algorithms. Even so, no Moving Picture format fulfills any of the qualifying requirements for linked open data.

Any dataset composed of Moving Pictures must explain in the description why the dataset is useful.

Music

<table>
<thead>
<tr>
<th>Format</th>
<th>Specification</th>
<th>Openness</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEI</td>
<td>Specifications [<a href="http://www.music-encoding.org/documentation/guidelines2013">http://www.music-encoding.org/documentation/guidelines2013</a>]</td>
<td>★★★(★★)</td>
</tr>
</tbody>
</table>

Service

<table>
<thead>
<tr>
<th>Format</th>
<th>Specification</th>
<th>Openness</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARQL</td>
<td>Specification [<a href="http://www.w3.org/TR/sparql11-service-description/">http://www.w3.org/TR/sparql11-service-description/</a>]</td>
<td>★★★(★★)</td>
</tr>
</tbody>
</table>

Still Image

Text

For-
HTML

Nowadays much data is available in HTML format on various sites. This may well be sufficient if the data is very stable and limited in scope. In some cases, it could be preferable to have data in a form easier to download and manipulate, but as it is cheap and easy to refer to a page on a website, it might be a good starting point in the display of data.

Typically, it would be most appropriate to use tables in HTML documents to hold data, and then it is important that the various data fields are displayed and are given IDs which make it easy to find and manipulate data. Yahoo has developed a tool ([http://developer.yahoo.com/yql/](http://developer.yahoo.com/yql/)) that can extract structured information from a website, and such tools can do much more with the data if it is carefully tagged.

Plain Text

Plain text documents (.txt) are very easy for computers to read. They generally exclude structural metadata from inside the document however, meaning that developers will need to create a parser that can interpret each document as it appears.

Some problems can be caused by switching plain text files between operating systems. MS Windows, Mac OS X and other Unix variants have their own way of telling the computer that they have reached the end of the line.

TEI

TEI documents include structural metadata as well as the readable text of a document. TEI documents that use URIs to reference controlled vocabularies can be considered forms of linked data. Such documents under an open license can earn [five stars](http://lab.linkeddata.deri.ie/2010/star-scheme-by-example/) of openness.

Text Document

Classic documents in formats like Word, ODF, OOXML, or PDF may be sufficient to show certain kinds of data—for example, relatively stable mailing lists or equivalent. It may be cheap to exhibit in, as often it is the format the data is born in. The format gives...
no support to keep the structure consistent, which often means that it is difficult to enter
data by automated means. Be sure to use templates as the basis of documents that will
display data for reuse, so it is at least possible to pull information out of documents.

It can also support the further use of data to use typography markup as much as possible
so that it becomes easier for a machine to distinguish headings (any type specified) from
the content and so on. Generally it is recommended not to exhibit in word processing
format, if data exists in a different format.

Linked Data Ontologies

This section contains material from Linked Data <http://linkeddata.org/> available
under the CC-BY-SA <http://creativecommons.org/licenses/by-sa/3.0/> license.

Linked Data is about using the Web to connect related data that wasn’t previously
linked, or using the Web to lower the barriers to linking data currently linked using
other methods. More specifically, Wikipedia defines Linked Data as “a term used to de-
scribe a recommended best practice for exposing, sharing, and connecting pieces of data,
information, and knowledge on the Semantic Web using URIs
<http://en.wikipedia.org/wiki/URI> and RDF

This section contains links to common Linked Data ontologies and schemas. For each
schema, we’ve included the abstract from the specification.

Annotation
Open Annotation

The Open Annotation Core Data Model
<http://www.openannotation.org/spec/core/> specifies an interoperable framework
for creating associations between related resources, annotations, using a methodology
that conforms to the Architecture of the World Wide Web. Open Annotations can easily
be shared between platforms, with sufficient richness of expression to satisfy complex
requirements while remaining simple enough to also allow for the most common use
cases, such as attaching a piece of text to a single web resource.

An Annotation is considered to be a set of connected resources, typically including a
body and target, where the body is somehow about the target. The full model supports
additional functionality, enabling semantic annotations, embedding content, selecting
segments of resources, choosing the appropriate representation of a resource and pro-
viding styling hints for consuming clients.

Digital Facsimile
Shared Canvas

The Shared Canvas data model <http://www.shared-canvas.org/datamodel/spec/>
specifies a linked data based approach for describing digital facsimiles of physical objects
in a collaborative fashion. It is intended for use in the cultural heritage domain, although
may be useful in other areas, and is designed around requirements derived from digi-
tized text-bearing objects such as medieval manuscripts. Instances of the data model are consumed by rendering platforms in order to understand the relationships between the constituent text, image, audio or other resources. These resources are associated with an abstract Canvas, or parts thereof, via Open Annotations and the Annotations are grouped and ordered in OAI-ORE Aggregations.
Organizations

Organizations act like publishing departments for datasets (for example, the Department of Health). This means that datasets can be published by and belong to a department instead of an individual user. Within organizations, admins can assign roles and authorisation its members, giving individual users the right to publish datasets from that particular organisation (e.g. Office of National Statistics).
5 ★ Open Data

Tim Berners-Lee, the inventor of the Web and Linked Data initiator, suggested a 5 star deployment scheme for Open Data. Here, we give examples for each step of the stars and explain costs and benefits that come along with it.

By Example ...
Below, we provide examples for each level of Tim's 5 star Open Data plan. The example data used throughout is 'the temperature forecast for Galway, Ireland for the next 3 days':

- ⭐ make your stuff available on the Web (whatever format) under an open license
- ⭐⭐ make it available as structured data (e.g., Excel instead of image scan of a table)
- ⭐⭐⭐ use non-proprietary formats (e.g., CSV instead of Excel)
- ⭐⭐⭐⭐ use URIs to denote things, so that people can point at your stuff
- ⭐⭐⭐⭐⭐ link your data to other data to provide context

Costs & Benefits ...

What are the costs & benefits of ★ Web data?

As a consumer ...

✔ You can look at it.
✔ You can print it.
✔ You can store it locally (on your hard drive or on an USB stick).
✔ You can enter the data into any other system.
✔ You can change the data as you wish.
✔ You can share the data with anyone you like.

As a publisher ...

✔ It's simple to publish.
✔ You do not have explain repeatedly to others that they can use your data.

“It's great to have the data accessible on the Web under an open license (such as PDDL, ODC-by or CC0), however, the data is locked-up in a document. Other than writing a custom scraper, it's hard to get the data out of the document."
What are the costs & benefits of ★★ Web data?

As a consumer, you can do all what you can do with ★ Web data and additionally:

✔ You can directly process it with proprietary software to aggregate it, perform calculations, visualise it, etc.
✔ You can export it into another (structured) format.

As a publisher ...

✔ It's still simple to publish.

“Splendid! The data is accessible on the Web in a structured way (that is, machine-readable), however, the data is still locked-up in a document. To get the data out of the document you depend on proprietary software.”

What are the costs & benefits of ★★★ Web data?

As a consumer, you can do all what you can do with ★★ Web data and additionally:

✔ You can manipulate the data in any way you like, without being confined by the capabilities of any particular software.

As a publisher ...

⚠ You might need converters or plug-ins to export the data from the proprietary format.
✔ It's still rather simple to publish.

“Excellent! The data is not only available via the Web but now everyone can use the data easily. On the other hand, it's still data on the Web and not data in the Web.”
What are the costs & benefits of ★★★★ Web data?

As a consumer, you can do all what you can do with ★★★★ Web data and additionally:

✔ You can link to it from any other place (on the Web or locally).
✔ You can bookmark it.
✔ You can reuse parts of the data.
✔ You may be able to reuse existing tools and libraries, even if they only understand parts of the pattern the publisher used.
△ Understanding the structure of an RDF "Graph" of data can be more effort than tabular (Excel/CSV) or tree (XML/JSON) data.
✔ You can combine the data safely with other data. URIs are a global scheme so if two things have the same URI then it's intentional, and if so that's well on it's way to being 5 star data!

As a publisher ...

✔ You have fine-granular control over the data items and can optimise their access (load balancing, caching, etc.)
✔ Other data publishers can now link into your data, promoting it to 5 star!
△ You typically invest some time slicing and dicing your data.
△ You'll need to assign URIs to data items and think about how to represent the data.
△ You need to either find existing patterns to reuse or create your own.

" Wonderful! Now it's data in the Web. The (most important) data items have a URI and can be shared on the Web. A native way to represent the data is using RDF, however other formats such as Atom can be converted/mapped, if required. "

http://5stardata.info/
What are the costs & benefits of ★★★★★ Web data?

As a consumer, you can do all what you can do with ★★★★ Web data and additionally:

✔ You can discover more (related) data while consuming the data.
✔ You can directly learn about the data schema.
⚠ You now have to deal with broken data links, just like 404 errors in web pages.
⚠ Presenting data from an arbitrary link as fact is as risky as letting people include content from any website in your pages. Caution, trust and common sense are all still necessary.

As a publisher ...

✔ You make your data discoverable.
✔ You increase the value of your data.
✔ You own organisation will gain the same benefits from the links as the consumers.
⚠ You'll need to invest resources to link your data to other data on the Web.
⚠ You may need to repair broken or incorrect links.

“Brilliant! Now it's data, in the Web linked to other data. Both the consumer and the publisher benefit from the network effect.”

See Also

- Ed Summers has provided a nice rendering of the 5 star scheme that
we're building upon here.

- If you have Open Data, you might want to use the **star badges** to flag your compliance level.
- **James G. Kim** provided a *Korean translation* of this site.
- **Christopher Gutteridge** has a *Linked Data crash course for programmers*.
- **The Open Data Institute helps you** certify your open data with the *open data certificate*.

Kudos to **Andy Seaborne** for pointing out the CSV bug, to **Kerstin Forsberg** for suggesting the 'data highlighting' in the 4/5 star examples, as well as to **Vassilios Peristeras** for proposing to explain not only the 'what' but also the 'why'. Thanks to **Egon Willighagen** for providing more details about benefits of one-star data. Additional contributions from **Christopher Gutteridge**. This site is brought to you by the EC FP7 Support Action LOD-Around-The-Clock (LATC).

---

Last update: 2012-04-03 by **Michael** | Code available via **GitHub**
Linked Data Basics for Techies

From OpenOrg

Contents

1 Introduction
   1.1 Intended Audience
   1.2 Please Feedback-- especially if something doesn't make sense!!!!
   1.3 Warning
   1.4 Alternatives
2 Why Should I Care about RDF?
   2.1 Structure
   2.2 Merging
   2.3 Why don't we use it for everything?
3 RDF Data
   3.1 RDF & Triples
   3.2 URI vs URL
      3.2.1 Example
   3.3 RDF Documents
      3.3.1 RDF+XML
      3.3.2 Turtle (and N3)
      3.3.3 RDFa
      3.3.4 Other Serialisations
   3.4 Namespaces
      3.4.1 Example 1 - RDF/XML
      3.4.2 Example 2: Turtle
      3.4.3 Example 3: N-Triples
      3.4.4 Some Common Namespaces
   3.5 Two relationships you must know
      3.5.1 rdfs:label
      3.5.2 rdf:type
         3.5.2.1 a
         3.5.2.2 Defining your own types
         3.5.2.3 Don't use types to encode data
         3.5.2.4 Don't use all the things of a type in a document to infer something
   3.6 Semantics (Schemas, OWL, Ontologies etc)
   3.7 Lists
4 Tools and Services
   4.1 prefix.cc
   4.2 t-d-b.org
   4.3 Triple Stores and SPARQL
   4.4 Validation
   4.5 Command Line Tools
Introduction

Intended Audience

This is intended to be a crash course for a techie/programmer who needs to learn the basics ASAP. It is not intended as an introduction for managers or policy makers (I suggest looking at Tim Berners-Lee's TED talks (http://www.google.co.uk/search?q=ted+berners-lee) if you want the executive summary).

It's primarily aimed at people who're tasked with creating RDF and don't have time to faff around. It will also be useful to people who want to work with RDF data. RDF is a data structure perfect for people creating mash-ups!

Please Feedback-- especially if something doesn't make sense!!!!

If you are new to RDF/Linked Data then you can help me!

I put a fair bit of effort into writing this, but I am too familiar with the field!

If you are learning for the first time and something in this guide isn't explained very well, please drop me a line so I can improve it. cjg@ecs.soton.ac.uk

Warning

Some things in this guide are deliberately over-simplified. It is intended to get you started really fast, rather than cover every facet of the subject.

Alternatives

If you don't like my way of explaining things, then there's other introductions out there;

- Linked Data Tutorial (http://www4.wiwiss.fu-berlin.de/bizer/pub/linkeddatatutorial/)
- Linked Data Book (http://linkeddatabook.com/editions/1.0/)
Why Should I Care about RDF?

At first glance it's not very easy to see why we might want yet another data format on top of XML, JSON, CSV etc. RDF is different and is better in some situations.

Structure

Tabular data: (SQL Databases, Excel, CSV, etc.) information is arranged in a strict grid. Adding and removing data is easy, but changing the shape of the table is a much higher cost.

Tree data: (JSON, XML.) Tree-data is easy to get your head around as you can worry about little bits at a time. It can still be tricky to modify the structure and merge data from multiple sources, especially if those sources were not designed with the merger in mind.

Graph data: (RDF). A graph is a list of relationships between things. This can any shape. This can be a bit more work to get your head around when coding, compared to the more limited structures, but ultimately it's more flexible. A set of relationships merged with a set of relationships is just a bigger set of relationships, so merging two RDF documents is trivial.

Merging

RDF uses globally unique identifiers for everything; the things we're talking about, the relationships, datatypes. This means two RDF graphs can be merged and there's no danger of having any confusion, eg. one dataset has a 'pitch' value for the frequency of a sound, and another for the angle of a roof. Doesn't happen in RDF as it's unambiguous. Of course, that makes if far more verbose, but TANSTAAFL (http://en.wikipedia.org/wiki/There_ain%27t_no_such_thing_as_a_free_lunch).

Because all RDF really is is a list of unambiguous relationships, you can combine two RDF graphs and just get a bigger list of relationships between things. No other formats in common use allow this.

Why don't we use it for everything?

It's just not suited for all cases. It's verbosity can be a point against it some situations, as can the flexibility. More flexible means that there's a higher amount of cognitive load on understanding data in RDF.

SPARQL databases are much less mature than SQL databases. They don't (yet) have all the little niceties we've become accustomed to, but they are improving. I still don't believe they are suited for managing records with access controls etc. This will certainly change in the next few years.

RDF Data

RDF & Triples
RDF is a way of structuring information to be very interoperable and extendable. The most simple unit of information is called a 'triple'. A triple consists of three parts;

1. the ID of a thing,
2. the ID of a property that relates that thing to another thing or a value
3. the ID of the thing it relates to OR a value, like text, a number or date.

For example:

```
<Person23> <hasDateOfBirth> "1969-05-23" .
<Person23> <name> "Marvin Fenderson" .
<Person23> <memberOf> <Group003> .
```

The first thing is called the **Subject**. The second is sometimes called a **Predicate,Property** or **Relation**, the last bit is the **Object**. If the last bit is a value rather than the ID of a thing it's called a **Literal**. ID's may represent absolutely anything, but we use web addresses for them. These are called **URIs** (note that URI and URL are slightly different. It can be confusing at first because http://webscience.org/person/6 refers to a person, not a webpage, but it's a very handy way to ensure that these IDs are globally unique.

One little caveat, a literal can have a datatype (like integer or string, also represented by a URI, but we still call this a "Triple" (yes that's dumb)).

The neat thing about this structure is that you can represent almost any other kind of data using it. It's not great at doing ordered lists of values.

**URI vs URL**

A URI represents a single concept or thing, but many URIs can represent the same thing.

If you resolve a URI it's considered good practice to return some useful triples about the concept the URI represents, but don't lose sleep over doing that -- it's an optional bonus feature of RDF.

All URLs are URIs. Not all URIs are URLs.

**URI**: Universal Resource Indicator - this identifies something uniquely.

**URL**: Universal Resource Location - this not only identifies something, but also describes where it is located.

**Example**

<http://dbpedia.org/resource/Julius_Caesar> is a URI for Julius Caesar.

<http://en.wikipedia.org/wiki/Julius_Caesar> is a URL (and therefore also a URI) for a web page about Julius Caesar.

There is no **URL** for Julius Caesar as you can't download him via the web as he's dead and also not a string of ones and zeros!

Two URIs may indicate the same concept, just like two URLs may return exactly the same document, or you
can have more than one name to address the same person. <http://www4.wiwiss.fu-berlin.de/gutendata/resource/people/Caesar_Julius_100_BC-44_BC> is another URI for Julius Caesar, created by a different organisation. You can choose if you treat two URIs as referring to the same concept or not. It depends on the problem you're trying to solve. There are no objective truths!

**RDF Documents**

**RDF+XML**

There are several ways of writing RDF triples into a file. The most common is called RDF+XML (which people often just called RDF). It usually looks something like this:

```xml
<foaf:Person rdf:about='http://webscience.org/person/7'>
   <foaf:name>Christopher Gutteridge</foaf:name>
</foaf:Person>
```

If you want to produce RDF+XML See this Guide (http://blogs.ecs.soton.ac.uk/webteam/2010/11/08/what-you-need-to-know-about-rdfxml/) . To parse RDF+XML just find and use a library, there's one in most popular languages!

This wiki uses a simple subset of RDF+XML for examples.

**Turtle (and N3)**

N3 is quite complicated so some bright person defined a cut-down version called Turtle which is really easy to read and write, but is sadly not as widely supported as RDF+XML.

Turtle looks something like this:

```turtle
<http://webscience.org/person/7> a foaf:Person ;
   foaf:name "Christopher Gutteridge" .
```

Note that this is NOT XML. The angle brackets just go around URIs which are not abbreviated using a prefix. (see later in this guide)

**RDFa**

RDFa is a way to embed triples into an HTML document. It can be confusing for beginners, but some software tools generate valid RDFa which is fine, but don't try to create it by hand until you get some experience!

- http://drupal.org/node/574624 - Drupal (a content management system) can output RDFa

To decode RDFa in an HTML page, just put the URL into http://graphite.ecs.soton.ac.uk/browsere/
Other Serialisations

There's one for JSON, and N-Triples which just writes out triples, one per line (and is a subset of Turtle).

Namespaces

For URIs it's common to define a bunch of related concepts in the same "namespace". A namespace is bit like a directory on a filesystem; it usually ends with either "/" and "#" and the IDs in the namespace generally don't have "/" or "#" in as that confuses things.

You will probably define your own namespace for your own concepts, such as your organisation's members, or the bus stops nearby, but for classes and predicates you'll often use other people's namespaces. In RDF+XML and Turtle it's common to use a namespace prefix to make things more readable. In RDF+XML you must use namespace prefixes for predicates. The following examples mean exactly the same thing:

Example 1 - RDF/XML

(in RDF+XML you have to use a namespace for the predicates to make them legal XML tags)

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description rdf:about="http://webscience.org/person/7">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person" />
    <foaf:name>Christopher Gutteridge</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

Example 2: Turtle

Same data as Example 1 (Turtle, but represents the same data as above). Turtle auto defines 'rdf:' so you don't have to (unlike RDF+XML where you always have to define it). An annoying quirk of turtle is that the value to the right of the prefix (eg. the 'bar' in foo:bar) must start with a letter, not a numerical digit, but if in doubt the prefix is a convenience, you can always just put full URIs in angle brackets instead.

```turtle
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix wsperson: <http://webscience.org/person/> .
wsperson:p7 foaf:name "Christopher Gutteridge" .
```

Example 3: N-Triples

Same data as Example 1 and 2.

```turtle
```

See also the prefix.cc service listed in the "Tools and Services" section of this guide.
Some Common Namespaces

Here's a quick summary of the most common namespaces.

rdf (http://prefix.cc/rdf) - has the core parts of RDF, but usually you'll only see rdf:type.

rdfs (http://prefix.cc/rdfs) - used for making statements about predicates and classes, also has rdfs:label and rdfs:comment which are good basic ways of giving something a label and making comments about it.

owl (http://prefix.cc/owl) - used for making much more complex statements about predicates and classes. This is cool, but don't worry about it too much when you're just getting started. Also defines owl:sameAs to indicate two URIs represent the same thing (in your opinion).

dcters (http://prefix.cc/rdf) - Dublin Core terms. Very useful generic properties for making statements about resources -- who created them, when, who published it, title, description etc. An older version is called 'Dublin Core Elements' and this can be confusing. In general always use dcterms. Some people use "dc" as the abbreviation, but this is confusing as it's not obvious if it's dc-terms or dc-elements, so don't do that.

foaf (http://prefix.cc/foaf) - Friend of a Friend. This is good for describing the facts from a person or organisations 'profile', things like their email address, phone numbers, names, what groups they are a member of etc.

geo (http://prefix.cc/geo) - Allows you to specify a latitude & longitude of something. Even if it's a big thing, then you can still give a useful reference point to navigate to.

dbpedia (http://prefix.cc/dbpedia) - DBPedia defines a URI for the primary of every page of wikipedia. So http://dbpedia.org/resource/Southampton is a URI representing the city of Southampton.

Also: skos, sioc, void, dcat, geonames, doap, vcard, org, event, prog, bibo, aiiso, vann, scovo

For a list of the namespaces on prefix.cc see; http://prefix.cc/popular/all

Two relationships you must know

Almost every URI you define should have rdf:type and rdfs:label defined. Noe that those are different namespaces. It's very easy to type "rdf:label" by accident.

rdfs:label

Reading URIs can be hard. Use rdfs:label to give a human readable label to a URI. eg.

<http://id.southampton.ac.uk/building/59> rdfs:label "Building 59" .

If you want to be more international-friendly you can add a language code;

<http://id.southampton.ac.uk/building/59> rdfs:label "Building 59"@en .
<http://id.southampton.ac.uk/building/59> rdfs:label "Edificio 59"@es .
Actually, this is one of the real benefits of RDF; you can easily provide alternate labels in several languages. Or someone else can provide triples labelling a set of existing URIs in a new language.

**rdf:type**

The full URI for this relationship is http://www.w3.org/1999/02/22-rdf-syntax-ns#type

This relates a URI for a thing to a URI for a set of things. For example;

```
<http://id.southampton.ac.uk/building/59>
rdf:type <http://vocab.deri.ie/rooms#Building> .
```

The rdf:type of something gives you a broad indication of what properties something will have and how to work with it.

The rdf:type of something is usually referred to as "class" not "type". If you've done any set theory you can think of it as a set.

**a**

Turtle and SPARQL have an abbreviation of rdf:type which is just "a". As you use it so often this saves a far bit of typing, eg.

```
<http://id.ecs.soton.ac.uk/person/1248> a foaf:Person .
<http://id.southampton.ac.uk/building/59> a
<http://vocab.deri.ie/rooms#Building> .
```

**Defining your own types**

When designing a system I find it useful to try to give everything very broad, common types. This makes it possible for other people's tools to understand your data. If you do want to define more specific classes that's OK, but I suggest you also give each thing a broad class so tools can understand it, eg. If you were making a document about swimming you might have swim:Race and swim:Swimmer as types. I would strongly suggest using event:Event and foaf:Person as additional types.

In theory you could use the semantic definitions to say swim:Race rdfs:subClassOf event:Event . swim:Swimmer rdfs:subClassOf foaf:Person . and it's helpful to do so, but most apps don't bother trying to resolve such semantics, yet.

**Don't use types to encode data**

It's tempting to use rdf:type to indicate and set the thing belongs in, eg. <http://id.ecs.soton.ac.uk/person/1248>
<http://id.ecs.soton.ac.uk/person/1248> rdf:type myns:PeopleOverSixFoot .

In my experience this is a mistake. It makes it harder for someone else to get to grips with your data. Use rdf:type to indicate what properties something is likely to have. People probably have names, email addresses and parents. Buildings have a postcode, number of floors and occupants. PeopleWhoDontEatCheese are unlikely to have any properties that foaf:Person won't. The above data is better encoded as:
nb. I've used 'eatsCheese' as I find negative properties are always a bit more confusing. You could have had 'lactoseIntolerant' if that's what you meant, of course.

**Don't use all the things of a type in a document to infer something**

For example you might know that all the Buildings in http://data.southampton.ac.uk/dumps/places/2012-01-09/places.rdf happen to be buildings occupied by the University of Southampton. This is a potential for problems later as later this data may later contain buildings which are not explicitly occupied by the university.

Worse still, if you merged it with data from other sources, there's now no way to tell which buildings belong to the University of Southampton.

It's much better to use explicit than implicit data.

If it's data you are creating, add a triple to every building to *say* it's a building occupied by the University of Southampton. That way there can be no confusion later if it happens to also contain buildings which the university used to host conferences.

Note that the example file given doesn't (yet) follow my own advice, I'm still learning too!

**Semantics (Schemas, OWL, Ontologies etc)**

The semantic bit of the semantic web is that if you resolve the URI for a class or predicate you often get back some rdfs: and/or owl: describing what it means, and some semantics about it. This lets you do clever reasoning like knowing that 'ancestor' is a transative property so the ancestor of an ancestor of X is therefore also an ancestor of X.

This is complicated, and not required to get started. If it confuses you, don't worry too much about it.

Purists insist you should write schemas, just ignore them until YOU find a need to write a schema (eventually it turns out to be useful, but there's no hurry).

**Lists**

You can do lists by saying something like,

```xml
<Person> <hasToDoList> <List0001> .
<List0001> <label> "Marvin's TODO List" .
<List0001> rdf:type rdf:Seq .
<List0001> rdf:_1 "Buy Milk" .
<List0001> rdf:_2 "Walk Dog" .
<List0001> rdf:_3 "Drink Milk" .
```
Tools and Services

prefix.cc

You can get a list of the standard abbreviations for common namespaces using http://prefix.cc/

If you're really lazy, you can get the stub of an XML document out of it, for example http://prefix.cc/foaf,dcterms,geo.rdf

t-d-b.org

This is described by http://thing-described-by.org/ and provides a quick and simple way to generate a URI for something by using a web page about that thing.

http://www.williamshatner.com/ - A URI which is a URL for a webpage about William Shatner

http://t-d-b.org/?http://www.williamshatner.com/ - A URI for William Shatner himself (the thing-described-by the URL after the ? mark)

Triple Stores and SPARQL

A triple store is a bit like an SQL database, but optimised to just import, store, and query a huge pile of triples. Triple stores are queried using a language called SPARQL.

They are funky because rather than deal with triples document by document you can query over any facets of the data in the "SPARQL Endpoint". If you have the staff resources to do so, it's good practice to provide a SPARQL endpoint, but don't lose sleep if you don't.

These are useful and powerful but not required to produce and work with RDF and Open Linked Data.

- Beginners Guide to SPARQL (http://www.pezholio.co.uk/2011/01/a-beginners-guide-to-sparqling-linked-data-part-1/)

Validation

It's quite easy to write valid RDF serialisations, but you can check them using the W3C Validator Service (http://www.w3.org/RDF/Validator/) , what is also useful is to eyeball your data to make sure it looks sane. I use my own RDF Browser (http://graphite.ecs.soton.ac.uk/browser/) but you may prefer others. The Graphite Browser will show triples from any of N3, RDF+XML and RDFa.

Command Line Tools

There's a C-Library and unix/linux command line tool called "rapper" which can parse and validate various formats.
Guide to how to produce datasets

Well, this website is for that!

Other sites you should know about

Semantic Overflow

http://semanticoverflow.com/

Allows you to ask questions about this stuff, and see existing questions and answers.

CKAN

http://ckan.org/

This site is a comprehensive index of sources of data sets on the web (many, but not all, in RDF). You should consider registering your datasets if you want other people to find them.

GetTheData.org

This site is used by people to ask where to find open data: http://getthedata.org/

Open Linked Data

Tim BL, inventor of the web, says this is cool and you should be doing it. See if you agree...

A nice explanation of "5 star linked data" is available at http://5stardata.info/

Linked Data

What is Linked Data? It's when you either:

- Use URIs for Subjects and/or Objects from other peoples datasets.
- Use owl:sameAs to link your identifiers to other peoples.

This lets people do really cool mashups using data from multiple sources. There's over 100 known sites in the world who link to other dataatasets. See The Linked Data Cloud (http://richard.cyganiak.de/2007/10/lod/) to see how they interlink.

It doesn't have to be RDF, but it usually is.
Open Data

Open data is data with an open license (http://www.opendefinition.org/) which makes it easy for people to reuse it with confidence, and available online, to all, without restrictions.

Open Linked Data

Is the combination of the two, obviously.

How does that relate to RDF

RDF is designed to be returned when resolving a URI, so it's ideal for linked data. It doesn't have to be Open, any more than XML does.

Closed Linked Data

It can be potentially useful to use the linked data techniques on confidential data. You can use it to pull information from many databases in a company and perform queries for "business intelligence" tasks.

If can also be useful to link private data to public data. eg. a student lecture timetable is not a public document, but each lecture is associated with a module, a course, a lecturer and a room. All of which may well have open data available about them. An app consuming the private timetable can augment it with links to this open data.

Retrieved from "http://openorg.ecs.soton.ac.uk/wiki/Linked_Data_Basics_for_Techies"

- This page was last modified on 4 April 2012, at 12:20.
In addition to the classic “Web of documents” W3C is helping to build a technology stack to support a “Web of data,” the sort of data you find in databases. The ultimate goal of the Web of data is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network. The term “Semantic Web” refers to W3C’s vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.

**Linked Data**

The Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. RDF provides the foundation for publishing and linking your data. Various technologies allow you to embed data in documents (RDFa, GRDDL) or expose what you have in SQL databases, or make it available as RDF files.

**Vocabularies**

At times it may be important or valuable to organize data. Using OWL (to build vocabularies, or “ontologies”) and SKOS (for designing knowledge organization systems) it is possible to enrich data with additional meaning, which allows more people (and more machines) to do more with the data.

**Query**

Query languages go hand-in-hand with databases. If the Semantic Web is viewed as a global database, then it is easy to understand why one would need a query language for that data. SPARQL is the query language for the Semantic Web.

**Inference**

Near the top of the Semantic Web stack one finds inference — reasoning over data through rules. W3C work on rules, primarily through RIF and OWL, is focused on translating between rule languages and exchanging rules among different systems.

**Vertical Applications**

W3C is working with different industries — for example in Health Care and Life Sciences, eGovernment, and Energy — to improve collaboration, research and development, and innovation adoption through Semantic Web technology. For instance, by aiding decision-making in clinical research, Semantic Web technologies will bridge many forms of biological and medical information across institutions.
News

Extend your stay for the upcoming MultilingualWeb workshop! Join the LIDER Workshop 8-9 May, to discuss Wikipedia, Multilingual Analytics and Linked Data for Language Resources

26 March 2014
from Internationalization Activity Blog » w3cSemanticWeb

Aligned with the MultilingualWeb workshop (7-8 May, Madrid), the LIDER project is organizing a roadmapping workshop 8-9 May. The 8 May afternoon session will provide a keynote by Seth Grimes and also focus on the topic of Wikipedia for multilingual Web content. Via several panels including contributions from key Wikipedia engineers, we will discuss cross lingual analytics and intelligent multilingual content handling in Wikipedia. On 9 May, a 1/2 day session will focus on aspects of migrating language resources into linked data.

Mark your calendar now! A dedicated registration form including ways to contribute to the workshop agenda will be made available soon.

sandhawke
11 March 2014
from Decentralyze

Lots of people can’t seem to understand the relationship of the Web to the Internet. So I’ve come up with a simple analogy:

The Web is to the Internet as Beer is to Alcohol.

For some people, sometimes, they are essentially synonymous, because they are often encountered together. But of course they are fundamentally different things.

In this analogy, Email is like Wine: it’s the other universally popular use of the Internet/Alcohol.

But there are lots of other uses, too, somewhat more obscure. We could say the various chat protocols are the various Whiskeys. IRC is Scotch; XMPP is Bourbon.

gopher is obscure and obsolete, …. maybe melomel.

ssh is potato vodka.
I leave the rest to your imagination.

Note that the non-technician never encounters raw Internet, just like they never encounter pure alcohol. They wouldn’t know what it was if it stepped on their foot. Of course, chemists are quite familiar with pure alcohol, and network technicians and programmers are familiar with TCP, UDP, and IP.

The familiar smell of alcohol, that you can detect to some degree in nearly everything containing alcohol — that’s DNS.

Speaker deadline for Madrid MultilingualWeb Workshop is Friday, March 14

05 March 2014

from Internationalization Activity Blog » w3cSemanticWeb

We would like to remind you that the deadline for speaker proposals for the 7th MultilingualWeb Workshop (May 7–8, 2014, Madrid, Spain) is on Friday, March 14, at 23:59 UTC.

Featuring a keynote by Alolita Sharma (Director of Engineering, Wikipedia) and breakout sessions on linked open data and other critical topics, this Workshop will focus on the advances and challenges faced in making the Web truly multilingual. It provides an outstanding and influential forum for thought leaders to share their ideas and gain critical feedback.

While the organizers have already received many excellent submissions, there is still time to make a proposal, and we encourage interested parties to do so by the deadline. With roughly 200 attendees anticipated for the Workshop from a wide variety of profiles, we are certain to have a large and diverse audience that can provide constructive and useful feedback, with stimulating discussion about all of the presentations.

For more information and to register, please visit the Madrid Workshop Call for Participation.

sandhawke

27 February 2014
The world of computing has a huge problem with surveillance. Whether you blame the governments doing it or the whistleblowers revealing it, the fact is that consumer adoption and satisfaction is being inhibited by an entirely-justified lack of trust in the systems.

Here’s how the NSA can fix that, increase the safety of Americans, and, I suspect, redeem itself in the eyes of much of the country. It’s a way to act with honor and integrity, without betraying citizens, businesses, or employees. The NSA can keep doing all the things it feel it must to keep America safe (until/unless congress or the administration changes those rules) and by doing this additional thing it would be helping protect us all from the increasing dangers of cyber attacks. And it’s pretty easy.

The proposal is this: establish a voluntary certification system, where vendors can submit products and services for confidential NSA review. In concluding its review, the NSA would enumerate for the public all known security vulnerabilities of the item. It would be under no obligation to discover vulnerabilities. Rather, it would simply need to disclose to consumers all the vulnerabilities of which it happens know, at that time and on an ongoing basis, going forward.

Vendors could be charged a reasonable fee for this service, perhaps on the order 1% gross revenue for that product.

Crucially, the NSA would accept civil liability for any accidental misleading of consumers in its review statements. Even more important: the NSA chain of command from the top down to the people doing the review would accept criminal liability for any intentionally misleading statements, including omissions. I am not a lawyer, but I think this could be done easily by having the statements include sworn affidavits stating both their belief in these statements and their due diligence in searching across the NSA and related entities. I’m sure there are other options too.

If congress wants to get involved, I think it might be time to pass an anti zero day law, supporting NSA certification. Specifically, I’d say that anyone who knows of a security vulnerability in an NSA certified product must report it immediately to the NSA or the vendor (which must tell each other). 90 days after reporting it, the person who reported it would be free to tell anyone / everyone, with full whistleblower protection. Maybe this could just be done by the product TOS.

NSA certified products could still include backdoors and weaknesses of all sorts, but their existence would no longer be secret. In particular, if there’s an NSA back door, a cryptographic hole for which they believe they have the only key, they would have to disclose that.

That’s it. Dear NSA, can you do this please?

For the rest of you, if you work at the kind of company the Snowden documents reveal to have been compromised, the companies who somehow handle user data, would you support this? Would your company participate in the program, regaining user trust?
Alolita Sharma (Wikipedia) to deliver keynote at 7th Multilingual Web Workshop (May 7–8, 2014, Madrid)

24 February 2014
from Internationalization Activity Blog » w3cSemanticWeb

We are pleased to announce that Alolita Sharma, Director of Engineering for Internationalization and Localization at Wikipedia, will deliver the keynote at the 7th Multilingual Web Workshop, “New Horizons for the Multilingual Web,” in Madrid, Spain (7–8 May 2014).

With over 30 million articles in 286 languages as of January 1, 2014, Wikipedia has now become one of the largest providers of multilingual content in the world. Because of its user-generated and constantly changing content, many traditional processes for managing multilingual content on the web either do not work or do not scale well for Wikipedia. Alolita Sharma’s keynote will highlight Wikipedia’s diversity in multilingual user-generated content and the language technologies that Wikipedia has had to develop to support its unprecedented growth of content. She will also discuss the many challenges Wikipedia faces in providing language support for the mobile web.

The Multilingual Web Workshop series brings together participants interested in the best practices, new technologies, and standards needed to help content creators, localizers, language tools developers, and others address the new opportunities and challenges of the multilingual Web. It will provide for networking across communities and building connections.

Registration for the Workshop is free, and early registration is recommended since space at the Workshop is limited.

There is still opportunity for individuals to submit proposals to speak at the workshop. Ideal proposals will highlight emerging challenges or novel solutions for reaching out to a global, multilingual audience. The deadline for speaker proposals is March 14, but early submission is strongly encouraged. See the Call for Participation for more details.

This workshop is made possible by the generous support of the LIDER project, which will organize a roadmapping workshop on linked data and content analytics as one of the tracks at Multilingual Web Workshop.

MultilingualWeb-LT Working Group closed, ITS community continues in ITS IG

10 February 2014
The MultilingualWeb-LTWorking Group has been closed, since it successfully completed the work in its charter.

We thank the co-chairs, the editors, implementers and the Working Group for achieving the goal to publish Internationalization Tag Set (ITS) 2.0 as a W3C Recommendation, and for doing so ahead of the original schedule.

Work on enlarging the community around ITS, gathering feedback and requirements for future work will now continue in the ITS Interest Group.

Call for position statements: Linked Data for Language Technology
#LiderEU
03 February 2014
from Internationalization Activity Blog » w3cSemanticWeb

The LD4LT (Linked Data for Language Technology) Workshop will be held on 21 March, in Athens, Greece, aligned with the European Data Forum 2014. See the agenda.

The workshop is a free community event – there is no admission fee for participants, but registration is required.

You are encouraged to provide a title for a position statement in your registration form. This is a simple, short statement that summarizes your ideas / technologies / use cases related to Linked Data and Language Technology.

The meeting is supported by the LIDER project, the MultilingualWeb community, the NLP2RDF project, the Working Group for Open Data in Linguistics as well as the DBpedia Project.

As input to the discussion and the work of the LD4LT group, you may also want to fill in the first LIDER survey.

First public working draft of Encoding published
31 January 2014
from Internationalization Activity Blog » w3cSemanticWeb

The Internationalization Working Group has published a First Public Working Draft of Encoding.

While encodings have been defined to some extent, implementations have not always implemented
them in the same way, have not always used the same labels, and often differ in dealing with undefined
and former proprietary areas of encodings. This specification attempts to fill those gaps so that new
implementations do not have to reverse engineer encoding implementations of the market leaders and
existing implementations can converge.

This is a snapshot of the Encoding Living Standard, as of the date shown on the title page. No changes
have been made in the body of the W3C draft other than to align with W3C house styles. The primary
reason that W3C is publishing this document is so that HTML5 and other specifications may
normatively refer to a stable W3C Recommendation.

Register now for the 7th W3C MultilingualWeb Workshop, Madrid, 7-8 May
30 January 2014
from Internationalization Activity Blog » w3cSemanticWeb

Register early to ensure you get a place. Anyone may attend all sessions at no charge and the
W3C welcomes participation by both speakers and non-speaking attendees.

Since 2010 the W3C’s Multilingual Web Workshop series has become the preeminent venue for
discussion of the standards and technologies that define and enable multilingualism on the Web. The
7th Workshop, “New Horizons for the Multilingual Web,” will be held 7–8 May 2014 in Madrid, Spain.

The workshop brings together participants interested in the best practices, new technologies, and
standards needed to help content creators, localizers, language tools developers, and others address
the new opportunities and challenges of the multilingual Web. It will provide for networking across
communities and building connections.

We are particularly interested in speakers who are facing emerging challenges or who can
demonstrate novel solutions for reaching out to a global, multilingual audience. The deadline for
speaker proposals is March 14, but early submission is strongly encouraged.

This workshop is made possible by the generous support of the LIDER project, which will organize a
roadmapping workshop on linked data and content analytics as one of the tracks at Multilingual Web
Workshop.

See the Call for Participation and register online.

Building bridges: Tutorial, Linked Data for Language Technologies at LREC
2014 Conference #LiderEU
24 January 2014
from Internationalization Activity Blog » w3cSemanticWeb
Under the umbrella of the Lider project and the MutilingualWeb community, the tutorial on Linked Data for Language Technologies aims at building bridges between two communities. Experts in language resources and applications will learn how to work with technical building blocks of linked data (RDF, SPARQL, ...); how to build linked data lexicon representations using the LEMON model; and how to integrate natural language processing workflows using the RDF NIF format. The tutorial is part of the LREC 2014 conference. The presenters are key participants in the LIDER projects and in W3C community groups like OntoLex, Best Practices for Multilingual Linked Open Data, and Linked Data for Language Technology.

**TALKS AND APPEARANCES**

None. See full list of W3C Talks and Appearances.

**EVENTS**

None. See full list of W3C Events.
Linked Data

What is Linked Data?
The Semantic Web is a Web of Data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web technologies (RDF, OWL, SKOS, SPARQL, etc.) provides an environment where application can query that data, draw inferences using vocabularies, etc.

However, to make the Web of Data a reality, it is important to have the huge amount of data on the Web available in a standard format, reachable and manageable by Semantic Web tools. Furthermore, not only does the Semantic Web need access to data, but relationships among data should be made available, too, to create a Web of Data (as opposed to a sheer collection of datasets). This collection of interrelated datasets on the Web can also be referred to as Linked Data.

To achieve and create Linked Data, technologies should be available for a common format (RDF), to make either conversion or on-the-fly access to existing databases (relational, XML, HTML, etc). It is also important to be able to setup query endpoints to access that data more conveniently. W3C provides a palette of technologies (RDF, GRDDL, POWDER, RDFa, the upcoming R2RML, RIF, SPARQL) to get access to the data.

What is Linked Data Used For?
Linked Data lies at the heart of what Semantic Web is all about: large scale integration of, and reasoning on, data on the Web. Almost all applications listed in, say collection of Semantic Web Case Studies and Use Cases are essentially based on the accessibility of, and integration of Linked Data at various level of complexities.

Examples
A typical case of a large Linked Dataset is DBPedia, which, essentially, makes the content of Wikipedia available in RDF. The importance of DBPedia is not only that it includes Wikipedia data, but also that it incorporates links to other datasets on the Web, e.g., to Geonames. By providing those extra links (in terms of RDF triples) applications may exploit the extra (and possibly more precise) knowledge from other datasets when developing an application; by virtue of integrating facts from several datasets, the application may provide a much better user experience.

Learn More
Tim Berners-Lee’s note on Linked Data gives a succinct description of the Linked Data principles. The Semantic Web community also maintains a list of books on a W3C Wiki page. Some of those books are...
introductory in nature while others are conference proceedings or textbook that address more advanced topics. Details of recent and upcoming Semantic Web related talks, given by the W3C Staff, the staff of the W3C Offices, and members of the W3C Working Groups are available separately; the slides are usually publicly available. The W3C also maintains a collection of Semantic Web Case Studies and Use Cases that show how Semantic Web technologies, including Linked Data, are used in practice.

Recent Press

26 March
Digital Journal
“The Apache Software Foundation Celebrates Document Freedom Day 2014”

20 March
BusinessWire
“Pharmaceutical and IT Communities Collaborate on OASIS Clinical Trial Data Standard for Content Management Systems”

12 March
The Kathimerini
“Η ελληνική συμμετοχή στη δημιουργία του Διαδικτύου (The Greek participation in the creation of the Internet)”

12 March
Mail Online
“Web creator Berners-Lee calls for digital 'Magna Carta' to enshrine rights of users across the world on 25th anniversary”

Current Status of Specifications

Learn more about the current status of specifications related to:

- RDF
- RDF Best Practices
- RDFa
- RDF Relationship to Other Formats
- GRDDL
- POWDER
- Semantic Annotation for WSDL and XML Schema
- Provenance
- RDB2RDF
- Linked Data

These W3C Groups are working on the related specifications:

- Government Linked Data Working Group
- Linked Data Platform (LDP) Working Group
- RDF Working Group
Inference

What is Inference?

Broadly speaking, inference on the Semantic Web can be characterized by discovering new relationships. On the Semantic Web, data is modeled as a set of (named) relationships between resources. “Inference” means that automatic procedures can generate new relationships based on the data and based on some additional information in the form of a vocabulary, e.g., a set of rules. Whether the new relationships are explicitly added to the set of data, or are returned at query time, is an implementation issue.

On the Semantic Web, the source of such extra information can be defined via vocabularies or rule sets. Both of these approaches draw upon knowledge representation techniques. In general, ontologies concentrate on classification methods, putting an emphasis on defining ‘classes’, ‘subclasses’, on how individual resources can be associated to such classes, and characterizing the relationships among classes and their instances. Rules, on the other hand, concentrate on defining a general mechanism on discovering and generating new relationships based on existing ones, much like logic programs, like Prolog, do. In the family of Semantic Web related W3C Recommendations RDFS, OWL, or SKOS are the tools of choice to define ontologies, whereas RIF has been developed to cover rule based approaches.

What Inference is Used For?

Inference on the Semantic Web is one of the tools of choice to improve the quality of data integration on the Web, by discovering new relationships, automatically analysing the content of the data, or managing knowledge on the Web in general. Inference based techniques are also important in discovering possible inconsistencies in the (integrated) data.

Examples

A simple example may help. The data set to be considered may include the relationship (Flipper isA Dolphin). An ontology may declare that “every Dolphin is also a Mammal”. That means that a Semantic Web program understanding the notion of “X is also Y” can add the statement (Flipper isA Mammal) to the set of relationships, although that was not part of the original data. One can also say that the new relationship was “discovered”. Another example is to express that fact that “if two persons have the same name, home page, and email address, then they are identical”. In this case, the “identity” of two resources can be discovered via inferencing.

Usage and techniques of ontologies and rules largely overlap. Very broadly speaking, ontologies optimize for taxonomic reasoning problems, and rule based systems optimize for reasoning problems
within the data. The difference is largely a matter of style, and criteria like available expertise, ease of adapting to existing data, tooling support, maturity and costs, etc., should be considered as far more important when trying to choose.

Learn More
The Semantic Web community maintains a list of books on a W3C Wiki page. Some of those books are introductory in nature while others are conference proceedings or textbook that address more advanced topics. Details of recent and upcoming Semantic Web related talks, given by the W3C Staff, the staff of the W3C Offices, and members of the W3C Working Groups are available separately; the slides are usually publicly available. The W3C also maintains a collection of Semantic Web Case Studies and Use Cases that show how Semantic Web technologies, including inference, is used in practice. Finally, the Semantic Web FAQ may also be of help in understanding the various concepts.

Current Status of Specifications
Learn more about the current status of specifications related to:

- RIF Rule Interchange Format
- OWL Web Ontology Language
- RDF

These W3C Groups are working on the related specifications:

- Government Linked Data Working Group
- Linked Data Platform (LDP) Working Group
- RDF Working Group
- RDFa Working Group
- Semantic Web Interest Group

Copyright © 2013 W3C ® (MIT, ERCIM, Keio, Beihang) Usage policies apply.
What is a Vocabulary?

On the Semantic Web, vocabularies define the concepts and relationships (also referred to as “terms”) used to describe and represent an area of concern. Vocabularies are used to classify the terms that can be used in a particular application, characterize possible relationships, and define possible constraints on using those terms. In practice, vocabularies can be very complex (with several thousands of terms) or very simple (describing one or two concepts only).

There is no clear division between what is referred to as “vocabularies” and “ontologies”. The trend is to use the word “ontology” for more complex, and possibly quite formal collection of terms, whereas “vocabulary” is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web.

What are Vocabularies Used For?

The role of vocabularies on the Semantic Web are to help data integration when, for example, ambiguities may exist on the terms used in the different data sets, or when a bit of extra knowledge may lead to the discovery of new relationships. Consider, for example, the application of ontologies in the field of health care. Medical professionals use them to represent knowledge about symptoms, diseases, and treatments. Pharmaceutical companies use them to represent information about drugs, dosages, and allergies. Combining this knowledge from the medical and pharmaceutical communities with patient data enables a whole range of intelligent applications such as decision support tools that search for possible treatments; systems that monitor drug efficacy and possible side effects; and tools that support epidemiological research.

Another type of example is to use vocabularies to organize knowledge. Libraries, museums, newspapers, government portals, enterprises, social networking applications, and other communities that manage large collections of books, historical artifacts, news reports, business glossaries, blog entries, and other items can now use vocabularies, using standard formalisms, to leverage the power of linked data.

It depends on the application how complex vocabularies they use. Some applications may decide not to use even small vocabularies, and rely on the logic of the application program. Some application may choose to use very simple vocabularies like the one described in the examples section below, and let a general Semantic Web environment use that extra information to make the identification of the terms. Some applications need an agreement on common terminologies, without any rigor imposed by a logic system. Finally, some applications may need more complex ontologies with complex reasoning.
procedures. It all depends on the requirements and the goals of the applications.

To satisfy these different needs, W3C offers a large palette of techniques to describe and define different forms of vocabularies in a standard format. These include RDF and RDF Schemas, Simple Knowledge Organization System (SKOS), Web Ontology Language (OWL), and the Rule Interchange Format (RIF). The choice among these different technologies depend on the complexity and rigor required by a specific application.

Examples
A general example may help. A bookseller may want to integrate data coming from different publishers. The data can be imported into a common RDF model, eg, by using converters to the publishers' databases. However, one database may use the term “author”, whereas the other may use the term “creator”. To make the integration complete, and extra definition should be added to the RDF data, describing the fact that the relationship described as “author” is the same as “creator”. This extra piece of information is, in fact, a vocabulary (or an ontology), albeit an extremely simple one.

In a more complex case the application may need a more detailed ontology as part of the extra information. This may include formal description on how authors are to be uniquely identified (eg, in a US setting, by referring to a unique social security number), how the terms used in this particular application relate to other datasets on the Web (eg, Wikipedia or geographic information), how the term “author” (or “creator”) can be related to terms like “editors”, etc.

Learn More
The Semantic Web community maintains a list of books on a W3C Wiki page. Some of those books are introductory in nature while others are conference proceedings or textbook that address more advanced topics. Details of recent and upcoming Semantic Web related talks, given by the W3C Staff, the staff of the W3C Offices, and members of the W3C Working Groups are available separately; the slides are usually publicly available. The W3C also maintains a collection of Semantic Web Case Studies and Use Cases that show how Semantic Web technologies, including vocabularies, are used in practice. Finally, the Semantic Web FAQ may also be of help in understanding the various concepts.

Recent Press
26 March
Digital Journal
“The Apache Software Foundation Celebrates Document Freedom Day 2014”

Current Status of Specifications
Learn more about the current status of specifications related to:

RDF
OWL Web Ontology Language
SKOS
Evaluation and Report Language (EARL)
Government Linked Data
RDF vocabularies

These W3C Groups are working on the related specifications:

Evaluation and Repair Tools Working Group
Government Linked Data Working Group
Linked Data Platform (LDP) Working Group
RDF Working Group
RDFa Working Group
Semantic Web Interest Group

Copyright © 2013 W3C ® (MIT, ERCIM, Keio, Beihang) Usage policies apply.
Query

On this page → what is topic • what is topic used for • examples • learn more • current status of specifications and groups

What is Query?
“Query” in the Semantic Web context means technologies and protocols that can programmatically retrieve information from the Web of Data.

What is Query Used For?
The Semantic Web is a Web of data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. RDF provides the foundation for publishing and linking your data. Various technologies allow you to embed data in documents (RDFa, GRDDL) or expose what you have in SQL databases, or make it available as RDF files.

However, just as relational databases or XML need specific query languages (SQL and XQuery, respectively), the Web of Data, typically represented using RDF as a data format, needs its own, RDF-specific query language and facilities. This is provided by the SPARQL query language and the accompanying protocols. SPARQL makes it possible to send queries and receive results, e.g., through HTTP or SOAP.

Technically, SPARQL queries are based on (triple) patterns. RDF can be seen as a set of relationships among resources (i.e., RDF triples); SPARQL queries provide one or more patterns against such relationships. These triple patterns are similar to RDF triples, except that one or more of the constituent resource references are variables. A SPARQL engine would returns the resources for all triples that match these patterns.

Using SPARQL consumers of the Web of Data can extract possibly complex information (i.e., existing resource references and their relationships) which are returned, for example, in a table format. This table can be incorporated into another Web page; using this approach SPARQL provides a powerful tool to build, for example, complex mash-up sites or search engines that include data stemming from the Semantic Web.

Examples
The short introduction on inferences includes a small example with a dataset containing the triple (Flipper isA Dolphin). Using SPARQL the user may issue a query of the form (Flipper isA ?species) where ?species denotes a variable. The query engine checks through the data and retrieves the value of Dolphin as a possible value for ?species which constitutes a possible answer to the query. The dataset may also contain the triple (Flipper isA Mammal) (e.g., as a result of an inference); in this case Mammal is also a possible value for ?species which should also be returned as a result of the
query. By providing several triple patterns, complex queries can be created and used by the application.

Learn More

The Semantic Web community maintains a list of books on a W3C Wiki page. Some of those books are introductory in nature while others are conference proceedings or textbook that address more advanced topics. Details of recent and upcoming Semantic Web related talks, given by the W3C Staff, the staff of the W3C Offices, and members of the W3C Working Groups are available separately; the slides are usually publicly available. The W3C also maintains a collection of Semantic Web Case Studies and Use Cases that show how Semantic Web technologies, including queries, are used in practice. Finally, the Semantic Web FAQ may also be of help in understanding the various concepts.

Recent Press

12 March
Semanticweb.com
“The Web Is 25 — And The Semantic Web Has Been An Important Part Of It”
4 February
InfoQ
“JSON-LD Reaches W3C Recommendation Status”
31 December
semanticweb.com
“Good-Bye 2013”

Current Status of Specifications

Learn more about the current status of specifications related to:

SPARQL

These W3C Groups are working on the related specifications:

Government Linked Data Working Group

Copyright © 2013 W3C ® (MIT, ERCIM, Keio, Beihang) Usage policies apply.
Vertical Applications

What are Vertical Applications?
Vertical applications is the term used at W3C to denote particular, generic application areas, specific communities, etc, that explore how W3C technologies (e.g., Semantic Web technologies) can help their operations, improve their efficiencies, provide better user experiences, etc. Some of these application areas may decide to form some sort a group at W3C to cooperate with other W3C members to explore these possibilities further. Examples for such vertical applications that have contact with W3C on different levels are health care and life sciences, social spaces, digital libraries, financial services, oil & gas exploration, or e-Government.

What are Vertical Applications Used For?
Apart from the inherent advantages for a particular vertical application area to get a better familiarity with a particular W3C technology, these groups also provide valuable feedbacks on the technologies themselves. Vertical applications may bring forward specific and sometimes highly non-trivial use cases, requirements in terms of, say, the expressivity of vocabulary or query languages, efficiency considerations for inferences, etc. Work on the second generations of languages like OWL2 or SPARQL1.1, or the work on R2RML, has been greatly motivated by the feedbacks provided by these vertical applications.

Examples
A good example is the The Health Care and Life Science Interest Group. It has been set up, in 2005, to explore the usability of Semantic Web technologies in area like drug discovery, patient care management and reporting, publication of scientific knowledge, drug approval procedures, etc. The group has produced several demonstrations, publications, organized workshops and conferences, and has succeeded in turning this particular application area into one of the most active user communities of Semantic Web technologies. Most major drug research companies, university labs and research center working in the area, and specialized tool providers participated in this work. The feedbacks provided by this group was also significant for Semantic Web technologies; as an example the definition of some of the OWL2 Profiles was strongly influenced by the ontologies and vocabularies developed by this community.

Learn More
The Semantic Web community maintains a list of books on a W3C Wiki page. Some of those books are introductory in nature while others are conference proceedings or textbook that address more advanced topics. Details of recent and upcoming Semantic Web related talks, given by the W3C Staff,
the staff of the W3C Offices, and members of the W3C Working Groups are available separately; the slides are usually publicly available. The W3C also maintains a collection of Semantic Web Case Studies and Use Cases that show how Semantic Web technologies, including different vertical application areas, is used in practice.

**Current Status of Specifications**

Learn more about the current status of specifications related to:

- Health Care and Life Sciences (Semantic Web)
- eGovernment

These W3C Groups are working on the related specifications:

- Government Linked Data Working Group
- Semantic Web Health Care and Life Sciences Interest Group

Copyright © 2013 W3C ® (MIT, ERCIM, Keio, Beihang) Usage policies apply.
Abstract

Linked Data Platform (LDP) defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Please see the Working Group's implementation report.

This W3C Recommendation was published by the Linked Data Platform Working Group. Discussions of this document are on public-ldp-comments@w3.org (subscribe, archives).

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 14 October 2005 W3C Process Document.
This specification is non-normative.

This specification describes the use of HTTP for accessing, updating, creating and deleting resources from servers that expose their resources as Linked Data. It provides clarifications and extensions of the rules of Linked Data [LINKED-DATA]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs, so that they can discover more things.

This specification discusses standard HTTP and RDF techniques used when constructing clients and servers that create, read, and write Linked Data Platform Resources. LDP Primer provides an entry-level introduction with many examples in the context of a fictional application. LDP Best Practices and Guidelines discusses best practices that you should use, and anti-patterns you should avoid, when constructing these clients and servers.

This specification defines a special type of Linked Data Platform Resource: a Container. Containers are very useful in building application models involving collections of resources, often homogeneous ones. For example, universities offer a collection of classes and a collection of faculty members, each faculty member teaches a collection of courses, and so on. This specification discusses how to work with containers. Resources can be added to containers using standard HTTP operations like POST (see section 5.2.3 HTTP POST).

The intention of this specification is to enable additional rules and layered groupings of rules as additional specifications. The scope is intentionally narrow to provide a set of key rules for reading and writing Linked Data that most, if not all, other specifications will depend upon and implementations will support.

This specification provides some approaches to deal with large resources. An extension to this specification provides the ability to break large resource representations into multiple paged responses [LDP-PAGING].

For context and background, it could be useful to read Linked Data Platform Use Case and Requirements [LDP-UCR] and section 6. Notable information from normative references.

2. Terminology

Terminology is based on W3C's Architecture of the World Wide Web [WEBARCH] and Hyper-text Transfer Protocol ([RFC7230], [RFC7231], [RFC7232]).

A relationship between two resources when one resource (representation) refers to the other resource by means of a URI [WEBARCH].

A program that establishes connections for the purpose of sending one or more HTTP requests [RFC7230].

A program that accepts connections in order to service HTTP requests by sending HTTP responses.

The terms "client" and "server" refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others.

HTTP enables the use of intermediaries to satisfy requests through a chain of connections. There are three common forms of HTTP intermediary: proxy, gateway, and tunnel. In some cases, a single intermediary might act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request. [RFC7230].

A HTTP resource whose state is represented in any way that conforms to the simple lifecycle patterns and conventions in section 4. Linked Data Platform Resources.

An LDRP whose state is fully represented in RDF, corresponding to an RDF graph. See also the term RDF Source from [rdf11-concepts].

An LDRP whose state is not represented in RDF. For example, these can be binary or text documents that do not have useful RDF representations.

A LDRP representing a collection of linked documents (RDF Document [rdf11-concepts] or information resources [WEBARCH]) that responds to client requests for creation, modification, and/or enumeration of its linked members and documents, and that conforms to the simple lifecycle patterns and conventions in section 5. Linked Data Platform Containers.

A LDP that defines a simple link to its contained documents (information resources) [WEBARCH].

An LDP that adds the concept of membership, allowing the flexibility of choosing what form its membership triples take, and allows membership to be any resources [WEBARCH]; not only documents.

A LDP that adds the concept of membership, allowing the flexibility of choosing what form its membership triples take, and allows membership to be any resources [WEBARCH]; not only documents.
An LDPC similar to a LDP-DC that is also capable of having members whose URIs are based on the content of its contained documents rather than the URIs assigned to those documents.

**Membership**

The relationship linking an LDPC and its member LDPRs, which can be different resources than its contained documents. The LDPC often assists with managing the membership triples, whether or not the LDPC's URI occurs in them.

**Membership triples**

A set of triples that lists an LDPC's members. A LDPC's membership triples all have one of the following patterns:

- `membership-constant-URI membership-predicate member-derived-URI`
- `member-derived-URI membership-predicate membership-constant-URI`

The difference between the two is simply which position member-derived-URI occupies, which is usually driven by the choice of `membership-predicate`. Most predicates have a natural forward direction inherent in their name, and existing vocabularies contain useful examples that read naturally in each direction. `ldp:member` and `dcterms:isPartOf` are representative examples.

Each linked container exposes properties (see section 5.2.1 General) that allow clients to determine which pattern it uses, what the actual `membership-predicate` and `membership-constant-URI` values are, and (for containers that allow the creation of new members) what value is used for the `member-derived-URI` based on the client's input to the creation process.

**Membership predicate**

The predicate of all an LDPC's membership triples.

**Containment**

The relationship binding an LDPC to LDPRs whose lifecycle it controls and is aware of. The lifecycle of the contained LDPR is limited by the lifecycle of the containing LDPC; that is, a contained LDPR cannot be created (through LDP-defined means) before its containing LDPC exists.

**Containment triples**

A set of triples, maintained by the LDPC, that lists documents created by the LDPC but not yet deleted. These triples always have the form: `(LDPC URI, ldp:contains, document-URI)`.

**Minimal-container triples**

The portion of an LDPC's triples that would be present when the container is empty. Currently, this definition is equivalent to all the LDPC's triples minus its containment triples, and minus its membership triples (if either are considered part of its state), but if future versions of LDP define additional classes of triples then this definition would expand to subtract out those classes as well.

**LDP-server-managed triples**

The portion of an LDP's triples whose behavior is constrained directly by this specification; for example, membership triples and containment triples. This portion of resources' content does not include constraints imposed outside of LDP, for example by other specifications that the server happens to support, or by server implementation decisions.

### 2.1 Conventions Used in This Document

The namespace for LDP is `http://www.w3.org/ns/ldp#`. Sample resource representations are provided in `text/turtle` format [turtle].

Commonly used namespace prefixes:

- `@prefix dc: <http://purl.org/dc/terms/>`.
- `@prefix foaf: <http://xmlns.com/foaf/0.1/>`.
- `@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns/>`.
- `@prefix ldp: <http://www.w3.org/ns/ldp#>`.
- `@prefix xsd: <http://www.w3.org/2001/XMLSchema#>`.

### 3. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words MAY, MUST, MUST NOT, RECOMMENDED, SHOULD, and SHOULD NOT are to be interpreted as described in [RFC2119].

The status of the sections of Linked Data Platform 1.0 (this document) is as follows:

- 1. Introduction: non-normative
- 2. Terminology: normative
- 3. Conformance: normative
- 4. Linked Data Platform Resources: normative
- 5. Linked Data Platform Containers: normative
- 6. Notable information from normative references: non-normative
- 7. HTTP Header Definitions: normative
- 8. Security Considerations: non-normative
- A. Acknowledgements: non-normative
- B. Change History: non-normative
- C.1 Normative references: normative
- C.2 Non-normative references: non-normative

A conforming LDP client is a conforming HTTP client [RFC7230] that follows the rules defined by LDP in section 4, Linked Data Platform Resources and also section 5, Linked Data Platform Containers.

A conforming LDP server is a conforming HTTP server [RFC7230] that follows the rules defined by LDP in section 4, Linked Data Platform Resources when it is serving LDPRs, and also section 5, Linked Data Platform Containers when it is serving LDPCs. LDP does not constrain its behavior when serving other HTTP resources.

### 4. Linked Data Platform Resources

#### 4.1 Introduction

This section is non-normative.
Linked Data Platform Resources (LDPRs) are HTTP resources that conform to the simple patterns and conventions in this section. HTTP requests to access, modify, create or delete LDPRs are accepted and processed by LDP servers. Most LDPRs are domain-specific resources that contain data for an entity in some domain, which could be commercial, governmental, scientific, religious, or other.

Some of the rules defined in this document provide clarification and refinement of the base Linked Data rules [LINKED-DATA]; others address additional needs.

The rules for Linked Data Platform Resources address basic questions such as:

- What resource representations should be used?
- How is optimistic collision detection handled for updates?
- What should client expectations be for changes to linked-to resources, such as type changes?
- How can the server make it easy for the client to create resources?
- How do I GET the representation of a large resource broken up into pages?

Additional non-normative guidance is available in the LDP Best Practices and Guidelines that addresses questions such as:

- What literal value types should be used?
- Are there some typical vocabularies that should be reused?
- What guidelines exist when interacting with LDPRs that are common but are not universal enough to specify normatively?

The following sections define the conformance rules for LDP servers when serving LDPRs.

LDP-RS's representations may be too big, one strategy is to break up the response representation into client consumable chunks called pages. A separate LDP specification outlines the conformance rules around pagination [LDP-PAGING].

A LDP server can manage two kinds of LDPRs, those resources whose state is represented using RDF (LDP-RS) and those using other formats (LDP-NR). LDP-RSs have the unique quality that their representation is based on RDF, which addresses a number of use cases from web metadata, open data models, machine processable information, and automated processing by software agents [rdf11-concepts]. LDP-NRs are almost anything on the Web today: images, HTML pages, word processing documents, spreadsheets, etc. and LDP-RSs hold metadata associated with LDP-NRs in some cases.

The LDP-NRs and LDP-RSs are simply sub-types of LDPRs, as illustrated in Fig. 2 Class relationship of types of Linked Data Platform Resources.

---

**4.2 Resource**

**4.2.1 General**

4.2.1.1 LDP servers **MUST** at least be HTTP/1.1 conformant servers [RFC7230].

4.2.1.2 LDP servers **MAY** host a mixture of LDP-RSs and LDP-NRs. For example, it is common for LDP servers to need to host binary or text resources that do not have useful RDF representations.

4.2.1.3 LDP server responses **MUST** use entity tags (either weak or strong ones) as response ETag header values, for responses that contain resource representations or successful responses to HTTP HEAD requests.

4.2.1.4 LDP servers exposing LDPRs **MUST** advertise their LDP support by exposing a HTTP Link header with a target URI of http://www.w3.org/ns/ldp#Resource, and a link relation type of type (that is, rel="type") in all responses to requests made to an LDPR's HTTP Request-URI [RFC5988].

Note: The HTTP Link header is the method by which servers assert their support for the LDP specification on a specific resource in a way that clients can inspect dynamically at run-time. This is **not** equivalent to the presence of a (subject-URI, rdf:type, ldp:Resource) triple in an LDP-RS. The presence of the header asserts that the server complies with the LDP specification’s constraints on HTTP interactions with LDPRs, that is it asserts that the resource has ETags, supports OPTIONS, and so on, which is not true of all Web resources.

Note: A LDP server can host a mixture of LDP-RSs and LDP-NRs, and therefore there is no implication that LDP support advertised on one HTTP Request-URI means that other resources on the same server are also LDPRs. Each HTTP Request-URI needs to be individually inspected, in the absence of outside information.
4.2.1.5 LDP servers MUST assign the default base-URI for [RFC3987] relative-URI resolution to be the HTTP Request-URI when the resource already exists, and to the URI of the created resource when the request results in the creation of a new resource.

4.2.1.6 LDP servers MUST publish any constraints on LDP clients’ ability to create or update LDPRs, by adding a Link header with an appropriate context URI, a link relation of http://www.w3.org/ns/ldp#constrainedBy, and a target URI identifying a set of constraints [RFC5988], to all responses to requests that fail due to violation of those constraints. For example, a server that refuses resource creation requests via HTTP PUT, POST, or PATCH would return this Link header on its 4xx responses to such requests. The same Link header MAY be provided on other responses. LDP neither defines nor constrains the representation of the link's target resource. Natural language constraint documents are therefore permitted, although machine-readable ones facilitate better client interactions. The appropriate context URI can vary based on the request's semantics and method; unless the response is otherwise constrained, the default (the effective request URI) SHOULD be used.

4.2.2 HTTP GET

4.2.2.1 LDP servers MUST support the HTTP GET method for LDPRs.

4.2.2.2 LDP servers MUST support the HTTP response headers defined in section 4.2.8 HTTP OPTIONS for the HTTP GET method.

4.2.3 HTTP POST

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes no new requirements for LDPRs.

Clients can create LDPRs via POST (section 5.2.3 HTTP POST) to a LDPC, via PUT (section 4.2.4 HTTP PUT), or any other methods allowed for HTTP resources. Any server-imposed constraints on LDPR creation or update must be advertised to clients.

4.2.4 HTTP PUT

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPRs.

4.2.4.1 If a HTTP PUT is accepted on an existing resource, LDP servers MUST replace the entire persistent state of the identified resource with the entity representation in the body of the request. LDP servers MAY ignore LDP-server-managed properties, and MAY ignore other properties such as dcterms:modified and dcterms:creator if they are handled specially by the server (for example, if the server overrides the value or supplies a default value). Any LDP servers that wish to support a more sophisticated merge of data provided by the client with existing state stored on the server for a resource MUST use HTTP PATCH, not HTTP PUT.

4.2.4.2 LDP servers SHOULD allow clients to update resources without requiring detailed knowledge of server-specific constraints. This is a consequence of the requirement to enable simple creation and modification of LDPRs.

4.2.4.3 If an otherwise valid HTTP PUT request is received that attempts to change properties the server does not allow clients to modify, LDP servers MUST fail the request by responding with a 4xx range status code (typically 409 Conflict). LDP servers SHOULD provide a corresponding response body containing information about which properties could not be persisted. The format of the 4xx response body is not constrained by LDP.

Non-normative note: Clients might provide properties equivalent to those already in the resource’s state, e.g. as part of a GET/update representation/PUT sequence, and those PUT requests are intended to work as long as the LDP-server-managed properties are identical on the GET response and the subsequent PUT request. This is in contrast to other cases like write-once properties that the server does not allow clients to modify once set; properties like this are under client and/or server control but are not constrained by LDP, so they are not LDP-server-managed triples.

4.2.4.4 If an otherwise valid HTTP PUT request is received that contains properties the server chooses not to persist, e.g. unknown content, LDP servers MUST respond with an appropriate 4xx range status code [RFC7231]. LDP servers SHOULD provide a corresponding response body containing information about which properties could not be persisted. The format of the 4xx response body is not constrained by LDP. LDP servers expose these application-specific constraints as described in section 4.2.1 General.

4.2.4.5 LDP clients SHOULD use the HTTP If-Match header and HTTP ETags to ensure it isn’t modifying a resource that has changed since the client last retrieved its representation. LDP servers SHOULD require the HTTP If-Match header and HTTP ETags to detect collisions. LDP servers MUST respond with status code 412 (Condition Failed) if ETags fail to match when there are no other errors with the request [RFC7232]. LDP servers that require conditional requests MUST respond with status code 428 (Precondition Required) when the absence of a precondition is the only reason for rejecting the request [RFC6585].

4.2.4.6 LDP servers MAY choose to allow the creation of new resources using HTTP PUT.

4.2.5 HTTP DELETE

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes no new blanket requirements for LDPRs.

Additional requirements on HTTP DELETE for LDPRs within containers can be found in section 5.2.5 HTTP DELETE.

4.2.6 HTTP HEAD

Note that certain LDP mechanisms rely on HTTP headers, and HTTP generally requires that HEAD responses include the same headers as GET responses. Thus, implementers should also carefully read sections 4.2.2 HTTP GET and 4.2.8 HTTP OPTIONS.

4.2.6.1 LDP servers MUST support the HTTP HEAD method.
4.2.7 HTTP PATCH

Per [RFC5789], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPRs.

Any server-imposed constraints on LDPR creation or update must be advertised to clients.

4.2.7.1 LDP servers that support PATCH MUST include an Accept-Patch HTTP response header [RFC5789] on HTTP OPTIONS requests, listing patch document media type(s) supported by the server.

4.2.8 HTTP OPTIONS

This specification imposes the following new requirements on HTTP OPTIONS for LDPRs beyond those in [RFC7231]. Other sections of this specification, for example PATCH, Accept-Post, add other requirements on OPTIONS responses.

4.2.8.1 LDP servers MUST support the HTTP OPTIONS method.

4.2.8.2 LDP servers MUST indicate their support for HTTP Methods by responding to a HTTP OPTIONS request on the LDPR's URL with the HTTP Method tokens in the HTTP response header Allow.

4.3 RDF Source

The following section contains normative clauses for Linked Data Platform RDF Source.

4.3.1 General

4.3.1.1 Each LDP RDF Source MUST also be a conforming LDP Resource as defined in section 4.2 Resource, along with the restrictions in this section. LDP clients MAY infer the following triple: one whose subject is the LDP-RS, whose predicate is rdf:type, and whose object is ldp:Resource, but there is no requirement to materialize this triple in the LDP-RS representation.

4.3.1.2 LDP-RSs representations SHOULD have at least one rdf:type set explicitly. This makes the representations much more useful to client applications that don't support inferencing.

4.3.1.3 The representation of a LDP-RS MAY have an rdf:type of ldp:RDFSource for Linked Data Platform RDF Source.

4.3.1.4 LDP servers MUST provide an RDF representation for LDP-RSs. The HTTP Request-URI of the LDP-RS is typically the subject of most triples in the response.

4.3.1.5 LDP-RSs SHOULD reuse existing vocabularies instead of creating their own duplicate vocabulary terms. In addition to this general rule, some specific cases are covered by other conformance rules.

4.3.1.6 LDP-RS predicates SHOULD use standard vocabularies such as Dublin Core [DC-TERMS], RDF [rdf11-concepts] and RDF Schema [rdf-schema], whenever possible.

4.3.1.7 In the absence of special knowledge of the application or domain, LDP clients MUST assume that any LDP-RS can have multiple rdf:type triples with different objects.

4.3.1.8 In the absence of special knowledge of the application or domain, LDP clients MUST assume that the rdf:type values of a given LDP-RS can change over time.

4.3.1.9 LDP clients SHOULD always assume that the set of predicates for a LDP-RS of a particular type at an arbitrary server is open, in the sense that different resources of the same type may not all have the same set of predicates in their triples, and the set of predicates that are used in the state of any one LDP-RS is not limited to any pre-defined set.

4.3.1.10 LDP servers MUST NOT require LDP clients to implement inferencing in order to recognize the subset of content defined by LDP. Other specifications built on top of LDP may require clients to implement inferencing [rdf11-concepts]. The practical implication is that all content defined by LDP must be explicitly represented, unless noted otherwise within this document.

4.3.1.11 A LDP client MUST preserve all triples retrieved from a LDP-RS using HTTP GET that it doesn’t change whether it understands the predicates or not, when its intent is to perform an update using HTTP PUT. The use of HTTP PATCH instead of HTTP PUT for update avoids this burden for clients [RFC5789].

4.3.1.12 LDP clients MAY provide LDP-defined hints that allow servers to optimize the content of responses. section 7.2 Preferences on the Prefer Request Header defines hints that apply to LDP-RSs.

4.3.1.13 LDP clients MUST be capable of processing responses formed by a LDP server that ignores hints, including LDP-defined hints.

4.3.2 HTTP GET

4.3.2.1 LDP servers MUST respond with a Turtle representation of the requested LDP-RS when the request includes an Accept header specifying text/turtle, unless HTTP content negotiation requires a different outcome [turtle].

Non-normative note: In other words, Turtle must be returned by LDP servers in the usual case clients would expect (client requests it) as well as cases where the client requests Turtle or other media type(s); Content negotiation results in a tie, and Turtle is one of the tying media types. For example, if the Accept header lists text/turtle as one of several media types with the highest relative quality factor (q= value), LDP servers must respond with Turtle. HTTP servers in general are not required to resolve ties in this way.
or to support Turtle at all, but LDP servers are. On the other hand, if Turtle is one of several requested media types, but another media type the server supports has a higher relative quality factor, standard HTTP content negotiation rules apply and the server (LDP or not) would not respond with Turtle.

4.3.2.2 LDP servers SHOULD respond with a text/turtle representation of the requested LDP-RS whenever the Accept request header is absent [turtle].

4.3.2.3 LDP servers MUST respond with a application/ld+json representation of the requested LDP-RS when the request includes an Accept header, unless content negotiation or Turtle support requires a different outcome [JSON-LD].

4.4 Non-RDF Source

The following section contains normative clauses for Linked Data Platform Non-RDF Source.

4.4.1 General

4.4.1.1 Each LDP Non-RDF Source MUST also be a conforming LDP Resource in section 4.2 Resource, LDP Non-RDF Sources may not be able to fully express their state using RDF [rdf11-concepts].

4.4.1.2 LDP servers exposing an LDP Non-RDF Source MAY advertise this by exposing a HTTP Link header with a target URI of http://www.w3.org/ns/ldp#NonRDFSource, and a link relation type of type (that is, rel="type") in responses to requests made to the LDP-NR's HTTP Request-URI [RFC5988].

5. Linked Data Platform Containers

5.1 Introduction

This section is non-normative.

Many HTTP applications and sites have organizing concepts that partition the overall space of resources into smaller containers. Blog posts are grouped into blogs, wiki pages are grouped into wikis, and products are grouped into catalogs. Each resource created in the application or site is created within an instance of one of these container-like entities, and users can list the existing artifacts within one. Containers answer some basic questions, which are:

1. To which URLs can I POST to create new resources?
2. Where can I GET a list of existing resources?
3. How do I get information about the members along with the container?
4. How can I ensure the resource data is easy to query?
5. How is the order of the container entries expressed? [LDP-PAGING]

This document defines the representation and behavior of containers that address these issues. There are multiple types of containers defined to support a variety of use cases, all that support a base set of capabilities. The contents of a container is defined by a set of triples in its representation (and state) called the containment triples that follow a fixed pattern. Additional types of containers allow for the set of members of a container to be defined by a set of triples in its representation called the membership triples that follow a consistent pattern (see the linked-to definition for the possible patterns). The membership triples of a container all have the same predicate, called the membership predicate, and either the subject or the object is also a consistent value – the remaining position of the membership triples (the one that varies) define the members of the container. In the simplest cases, the consistent value will be the LDPC resource's URI, but it does not have to be. The membership predicate is also variable and will often be a predicate from the server application vocabulary or the ldp:member predicate.

In LDP 1.0, there exists a way for clients to request responses that contain only partial representations of the containers. Applications may define additional means by which the response representations contain a filtered set of data and including (or excluding) additional details, those means are application-specific and not defined in this document.

This document includes a set of guidelines for creating new resources and adding them to the list of resources linked to a container. It goes on to explain how to learn about a set of related resources, regardless of how they were created or added to the container's membership. It also defines behavior when resources created using a container are later deleted; deleting containers removes membership information and possibly performs some cleanup tasks on unreferenced member resources.

The following illustrates a very simple container with only three members and some information about the container (the fact that it is a container and a brief title):

Request to http://example.org/c1/:

```
EXAMPLE 1
GET /c1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

```
EXAMPLE 2
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "8caab0784245148bfe98b38d5bd6b13"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD,PUT
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type", <http://www.w3.org/ns/ldp#Resource>; rel="type"
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

http://example.org/c1/
```
The preceding example is very straightforward: there are containment triples whose subject is the container, whose predicate is `ldp:contains`, and whose objects are the URIs of the contained resources, and there is no distinction between member resources and contained resources. A POST to this container will create a new resource and add it to the list of contained resources by adding a new containme

triple to the container. This type of container is called a Linked Data Platform Basic Container.

Sometimes you have to build on existing models incrementally, so you have fewer degrees of freedom in the resource model. In these situations, it can be useful to help clients map LDP patterns onto existing vocabularies, or to include members not created via the container; LDP Direct Containers meet those kinds of needs. Direct Containers allow membership triples to use a subject other than the container itself as the consistent membership value, and/or to use a predicate from an application’s domain model as the membership predicate.

Let’s start with a pre-existing domain resource for a person’s net worth, as illustrated immediately below, and then see how a Container resource can be applied in subsequent examples:

Request to `http://example.org/netWorth/nw1/`:

```text
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

```text
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "0f6b5bd8dc1f754a1238a53b1da34f6b"
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
<link href="http://www.w3.org/ns/ldp#Resource">; rel="type">
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.
<http://example.org/netWorth/nw1/> a o:NetWorth;
 o:netWorthOf <http://example.org/users/JohnZSmith>;
 o:asset <assets/a1>,<assets/a2>;
 o:liability <liabilities/l1>,<liabilities/l2>,<liabilities/l3>.
```

In the preceding example, there is a `rdf:type` of `o:NetWorth` indicating the resource represents an instance of a person’s net worth and a `o:netWorthOf` predicate indicating the associated person. There are two sets of same-subject, same-predicate triples; one for assets and one for liabilities. Existing domain-specific applications exist that depend on those types and predicates, so changing them incompatibly would be frowned upon.

It would be helpful to be able to use LDP patterns to manage the assets and liabilities-related triples. Doing so using a Basic Container would require duplicating much of the information with different types and predicates, which would be onerous for large resources. Direct Containers provide a middle ground, by giving LDP clients a way to map existing domain-specific resources to LDP’s types and interaction models. In this specific example, it would be helpful to be able to manage the assets and liabilities triples consistently, for example by using LDP containers. One way to do this is to create two containers, one to manage assets and another liabilities, and separate HTTP resources. Existing clients have no need to interact with those containers, whereas LDP-enabled clients now have container URLs that they can interact with. The existing resource remains unchanged so that existing clients continue to function normally. This is illustrated in the set of related examples, one example per HTTP resource, starting with the LDP-RS example from before:

Request to `http://example.org/netWorth/nw1/`:

```text
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

```text
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "0f6b5bd8dc1f754a1238a53b1da34f6b"
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
<link href="http://www.w3.org/ns/ldp#Resource">; rel="type">
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.
```
The structure of the net worth resource is completely unchanged, so any existing domain-specific applications continue to work without impact. LDP clients, on the other hand, have no way to understand that the asset and liability triples correspond in any way to LDP containers. For that, they need the next two resources.

The first container is a LDP Direct Container to manage assets. Direct Containers add the concept of membership and flexibility on how to specify the membership triples.

Request to http://example.org/netWorth/nw1/assets/:

```
EXAMPLE 7
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

```
EXAMPLE 8
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "6df36eef2631a795bfe9ab76760ea75"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
     <http://www.w3.org/ns/ldp#Resource>; rel="type"
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology#>.
<http://example.org/netWorth/nw1/assets/>
  a ldp:DirectContainer;
  dcterms:title "The assets of JohnZSmith";
  ldp:membershipResource <http://example.org/netWorth/nw1/>;
  ldp:hasMemberRelation o:asset;
  ldp:contains <a1>, <a2>.
```

In the preceding asset container, the consistent membership value (membership-constant-URI, still in the subject position) is not the container itself – it is the (separate) net worth resource. The membership predicate is o:asset, from the domain model. A POST of an asset representation to the asset container will create a new asset and add it to net-worth's list of assets by adding a new membership triple to the resource and a containment triple to the container.

The second container is a LDP Direct Container to manage liabilities.

Request to http://example.org/netWorth/nw1/liabilities/:

```
EXAMPLE 9
GET /netWorth/nw1/liabilities/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

Response:

```
EXAMPLE 10
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "9f50da01f792281ddcfebe978372d07"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology#>.
<http://example.org/netWorth/nw1/liabilities/>
  a ldp:DirectContainer;
  dcterms:title "The liabilities of JohnZSmith";
  ldp:membershipResource <http://example.org/netWorth/nw1/>;
  ldp:hasMemberRelation o:liability;
  ldp:contains <l1>, <l2>, <l3>.
```

The preceding liability container is completely analogous to the asset container above, simply managing liabilities instead of assets and using the o:liability predicate.

To add another liability, a client would POST something like this to the liability container:

Request to http://example.org/netWorth/nw1/liabilities/:
Response:

Assuming the server successfully processes this request and assigns the URI `<http://example.org/netWorth/nw1/liabilities/l4>` to the newly created liability resource, at least two new triples would be added.

1. In the net worth resource, `<http://example.org/netWorth/nw1/>` o:liability `<liabilities/l4>`

You might wonder why we chose to create two new containers instead of making `<http://example.org/netWorth/nw1/>` itself a container. A single net worth container would be a fine design if `<http://example.org/netWorth/nw1/>` had only assets or only liabilities (basically: only a single predicate to manage), but since it has separate predicates for assets and liabilities an ambiguity arises: it is unspecified whether a client's creation request (POST) should add a new o:asset or o:liability triple. Having separate `<http://example.org/netWorth/nw1/assets/>` and `<http://example.org/netWorth/nw1/liabilities/>` containers allows both assets and liabilities to be created and linked to the net-worth resource using the appropriate predicate. Similar ambiguities arise if the client wishes to list the members and/or contained resources.

Continuing in the multiple containers direction, we will now extend our net worth example to add a container for advisors (people) that have managed the assets and liabilities. We have decided to identify these advisors with URLs that contain a fragment (hash) to represent these real-world resources, not the documents that describe them.

Request to `<http://example.org/netWorth/nw1/>`:

Response:

To handle this type of indirection, the triple with predicate of `ldp:insertedContentRelation` and object of `foaf:primaryTopic` informs clients that when POSTing to this container, they need to include a triple of the form (`<>`, `foaf:primaryTopic`, `topic-URI`) to inform the server which URI to use (`topic-URI`) in the new membership triple.

This type of container is referred to as a LDP Indirect Container. It is similar to an LDP Direct Container but it provides an indirection to add (via a create request) as a member any resource, including a URI representing a real-world object. Create requests to an LDP Direct Containers can only add information resources [WEBARCH] - the documents they create - as members.

To add another advisor, a client would POST something like this to the advisors container:

```example 11
POST /netWorth/nw1/liabilities/ HTTP/1.1
Host: example.org
Accept: text/turtle
Content-Type: text/turtle
Content-Length: 63

@prefix o: <http://example.org/ontology#>.
<>
  a o:Liability.
  # plus any other properties that the domain says liabilities have
```

```example 12
HTTP/1.1 201 Created
Location: http://example.org/netWorth/nw1/liabilities/l4
Date: Thu, 12 Jun 2014 19:56:13 GMT
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
     <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

```example 13
GET /netWorth/nw1/ HTTP/1.1
Host: example.org
Accept: text/turtle
```

```example 14
HTTP/1.1 200 OK
Content-Type: text/turtle
Date: Thu, 12 Jun 2014 18:26:59 GMT
ETag: "e8d129f45ac13848fd56213844a32b49"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
     <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: GET,OPTIONS,HEAD,PUT,DELETE
Transfer-Encoding: chunked

@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix o: <http://example.org/ontology#>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<http://example.org/netWorth/nw1/>
  a o:NetWorth;
  o:netWorthOf <http://example.org/users/JohnZSmith>;
  o:advisor `<advisors/bob#me>`,# URI of a person
  `<advisors/marsha#me>`.

<advisors/>
  a ldp:IndirectContainer;
  dcterms:title "The asset advisors of JohnZSmith";
  ldp:membershipResource <>;
  ldp:hasMemberRelation o:advisor;
  ldp:insertedContentRelation foaf:primaryTopic;
  ldp:contains `<advisors/bob>`,# URI of a document a.k.a. an information resource
  `<advisors/marsha>`,# describing a person
```
Request to `http://example.org/netWorth/nw1/advisors/`:

**EXAMPLE 15**

```plaintext
POST /netWorth/nw1/advisors/ HTTP/1.1
Host: example.org
Accept: text/turtle
Content-Type: text/turtle
Slug: george
Content-Length: 72

@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix o: <http://example.org/ontology#>.
<>
  a o:Advisor;
  foaf:primaryTopic <>.
# plus any other properties that the domain says advisors have
```

Response:

**EXAMPLE 16**

```plaintext
HTTP/1.1 201 Created
Location: http://example.org/netWorth/nw1/advisors/george
Date: Thu, 12 Jun 2014 19:56:13 GMT
Link: <http://www.w3.org/ns/ldp#RDFSource>; rel="type",
     <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

Assuming the server successfully processes this request and assigns the URI `http://example.org/netWorth/nw1/advisors/george` to the newly created advisor resource, at least two new triples would be added.

1. In the net worth resource, `<http://example.org/netWorth/nw1/> o:advisor <advisors/george#me>`
2. In the advisors container, `<http://example.org/netWorth/nw1/advisors/> ldp:contains <george>`

In summary, **Fig. 3 Class relationship of types of Linked Data Platform Containers** illustrates the LDP-defined container types: Basic, Direct, and Indirect, along with their class relationship to types of LDPRs.

Fig. 3 Class relationship of types of Linked Data Platform Containers

The following table illustrates some differences between membership and containment triples. For details on the normative behavior, see appropriate sections below.

<table>
<thead>
<tr>
<th>Completed Request</th>
<th>Membership</th>
<th>Effects</th>
<th>Containment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPR created in LDP-BC</td>
<td>New triple: (LDP:Container, ldp:contains, LDPR)</td>
<td>Same</td>
<td>New triple: (LDP:Container, ldp:contains, LDPR)</td>
</tr>
<tr>
<td>LDPR created in LDP-DC</td>
<td>New triple links LDP-Resource to created LDPR. LDP-Resource URI may be same as LDP-Container URI</td>
<td>New triple: (LDP:Container, ldp:contains, LDPR)</td>
<td>New triple: (LDP:Container, ldp:contains, LDPR)</td>
</tr>
<tr>
<td>LDPR created in LDP-IC</td>
<td>New triple links LDP-Resource to content indicated URI</td>
<td>New triple: (LDP:Container, ldp:contains, LDPR)</td>
<td>(LDP:Container, ldp:contains, LDPR) triple is removed</td>
</tr>
<tr>
<td>LDPR is deleted</td>
<td>Membership triple may be removed</td>
<td>Triples of form (LDP:Container, ldp:contains, LDPR) and contained LDPRs may be removed</td>
<td></td>
</tr>
<tr>
<td>LDPC is deleted</td>
<td>Triples and member resources may be removed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1.1 Retrieving Only Minimal-Container Triples

The representation of a container that has many members will be large. There are several important cases where clients need to access only the subset of the container's properties that are unrelated to member resources and unrelated to contained documents, for example to determine the membership triple pattern and membership predicate of an LDP-DC. LDP calls these minimal-container triples, because they are what remains when the container has zero members and zero contained resources. Since retrieving the whole container representation to get this information may be onerous for clients and cause unnecessary overhead on servers, we define a way to retrieve only these property values (and more generally, a way to retrieve only the subset of properties used to address other major clusters of use cases). LDP adds parameters to the HTTP Prefer header's `return=representation` preference for this (see section 7.2 Preferences on the Prefer Request Header for details).

The example listed here only shows a simple case where few minimal-container triples are retrieved. In real world situations more complex cases are likely, such as those that add other predicates to containers, for example providing validation information and associating SPARQL endpoints. [sparql11-query]

Here is an example requesting the minimal-container triples of a container identified by the URL `http://example.org/container1/`.
LDP recommends using PATCH to update these properties, if necessary. It provides no facility for updating them via PUT without replacing the entire container's state.

5.2 Container

The following section contains normative clauses for Linked Data Platform Container.

5.2.1 General

The Linked Data Platform does not define how clients discover LDPCs.

5.2.1.1 Each Linked Data Platform Container MUST also be a conforming Linked Data Platform RDF Source. LDP clients MAY infer the following triple: one whose subject is the LDPC, whose predicate is \texttt{rdf:type}, and whose object is \texttt{ldp:RDFSource}, but there is no requirement to materialize this triple in the LDPC representation.

5.2.1.2 The representation of a LDPC MAY have an \texttt{rdf:type} of \texttt{ldp:Container} for Linked Data Platform Container. Non-normative note: LDPCs might have additional types, like any LDP-RS.

5.2.1.3 LDPC representations SHOULD NOT use RDF container types \texttt{rdf:Bag}, \texttt{rdf:Seq} or \texttt{rdf:List}.

5.2.1.4 LDP servers exposing LDPCs MUST advertise their LDP support by exposing a HTTP \texttt{Link} header with a target URI matching the type of container (see below) the server supports, and a link relation type of \texttt{type} (that is, \texttt{rel="type"}) in all responses to requests made to the LDPC's HTTP Request-URI. LDP servers MAY provide additional HTTP \texttt{Link: rel="type"} headers. The notes on the corresponding LDPR constraint apply equally to LDPCs.

Valid container type URIs for \texttt{rel="type"} defined by this document are:

- \texttt{http://www.w3.org/ns/ldp#BasicContainer} - for LDP Basic Containers
- \texttt{http://www.w3.org/ns/ldp#DirectContainer} - for LDP Direct Containers
- \texttt{http://www.w3.org/ns/ldp#IndirectContainer} - for LDP Indirect Containers

5.2.1.5 LDP servers SHOULD respect all of a client's LDP-defined hints, for example which subsets of LDP-defined state the client is interested in processing, to influence the set of triples returned in representations of a LDPC, particularly for large LDPCs. See also [LDP-PAGING].

5.2.2 HTTP GET

Per section 4.2.2 HTTP GET the HTTP GET method is required and additional requirements can be found in section 5.2.1 General.

5.2.3 HTTP POST

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on creation or update MUST be advertised to clients.

5.2.3.1 LDP clients SHOULD create member resources by submitting a representation as the entity body of the HTTP POST to a known LDPC. If the resource was created successfully, LDP servers MUST respond with status code 201 (Created) and the Location header set to the new resource's URL. Clients shall not expect any representation in the response entity body on a 201 (Created) response.

5.2.3.2 When a successful HTTP POST request to a LDPC results in the creation of a LDPR, a containment triple MUST be added to the state of
the LDPC whose subject is the LDPC URI, whose predicate is `ldp:contains` and whose object is the URI for the newly created document (LDPR). Other triples may be added as well. The newly created LDPR appears as a contained resource of the LDPC until the newly created document is deleted or removed by other methods.

5.2.3.3 LDP servers MAY accept an HTTP POST of non-RDF representations (LDP-NRs) for creation of any kind of resource, for example binary resources. See the Accept-Post section for details on how clients can discover whether a LDP supports this behavior.

5.2.3.4 LDP servers that successfully create a resource from a RDF representation in the request entity body MUST honor the client's requested interaction model(s). If any requested interaction model cannot be honored, the server MUST fail the request.

- If the request header specifies a LDPR interaction model, then the server MUST handle subsequent requests to the newly created resource's URI as if it is a LDPR. When the server treats the resource as a LDPR, then clients can only depend upon behaviors defined by LDPRs, even if the content contains an `rdf:type` triple indicating a type of LDPC. That is, if the server's statement conflicts with the content's, the server's statement wins.
- If the request header specifies an LDP interaction model, then the server MUST handle subsequent requests to the newly created resource's URI as if it is a LDPC.
- This specification does not constrain the server's behavior in other cases.

Clients use the same syntax, that is HTTP Link headers, to specify the desired interaction model when creating a resource as servers use to advertise it on responses.

Note: A consequence of this is that LDPCs can be used to create LDPCs, if the server supports doing so.

Non-normative note: LDP assumes that the interaction model of a resource is fixed when the resource is created, and this is reflected in the language of the bullets. If an implementation were to extend LDP by allowing the interaction model to vary after creation, that is viewed as a compatible extension to LDP and the statements above would constrain the default interaction model rather than saying that no other behavior is possible.

5.2.3.5 LDP servers that allow creation of LDP-RSs via POST MUST allow clients to create new members by enclosing a request entity body with a `Content-Type` request header whose value is `text/turtle` [turtle].

5.2.3.6 LDP servers SHOULD use the `Content-Type` request header to determine the request representation's format when the request has an entity body.

5.2.3.7 LDP servers creating a LDP-RS via POST MUST interpret the null relative URI for the subject of triples in the LDP-RS representation in the request entity body as identifying the entity in the request body. Commonly, that entity is the model for the "to be created" LDPR, so triples whose subject is the null relative URI result in triples in the created resource whose subject is the created resource.

5.2.3.8 LDP servers SHOULD assign the URI for the resource to be created using server application specific rules in the absence of a client hint.

5.2.3.9 LDP servers SHOULD allow clients to create new resources without requiring detailed knowledge of application-specific constraints. This is a consequence of the requirement to enable simple creation and modification of LDPRs. LDP servers expose these application-specific constraints as described in section 4.2.1 General.

5.2.3.10 LDP servers MAY allow clients to suggest the URI for a resource created through POST using the HTTP Slug header as defined in [RFC5988]. LDP adds no new requirements to this usage, so its presence functions as a hint to the server providing a desired string to be incorporated into the server's final choice of resource URI.

5.2.3.11 LDP servers that allow member creation via POST SHOULD NOT re-use URIs.

5.2.3.12 Upon successful creation of an LDP-NR (HTTP status code of 201-Created and URI indicated by Location response header), LDP servers MAY create an associated LDP-RS to contain data about the newly created LDP-NR. If a LDP server creates this associated LDP-RS, it MUST include an `Accept-Post` response header on HTTP OPTIONS responses, listing POST request media type(s) supported by the server. LDP only specifies the use of POST for the purpose of creating new resources, but a server can accept POST requests with other semantics. While "POST to create" is a common interaction pattern, LDP clients are not guaranteed, even when making requests to a LDP server, that every successful POST request will result in the creation of a new resource; they must rely on out of band information for knowledge of which POST requests, if any, will have the "create new resource" semantics. This requirement on LDP servers is intentionally stronger than the one levied in the header registration; it is unrealistic to expect all existing resources that support POST to suddenly return a new header or for all new specifications constraining POST to be aware of its existence and require it, but it is a reasonable requirement for new specifications such as LDP.

5.2.3.13 LDP servers that support POST MUST include an Accept-Post response header on HTTP OPTIONS responses, listing POST request media type(s) supported by the server. LDP only specifies the use of POST for the purpose of creating new resources, but a server can accept POST requests with other semantics. While "POST to create" is a common interaction pattern, LDP clients are not guaranteed, even when making requests to a LDP server, that every successful POST request will result in the creation of a new resource; they must rely on out of band information for knowledge of which POST requests, if any, will have the "create new resource" semantics. This requirement on LDP servers is intentionally stronger than the one levied in the header registration; it is unrealistic to expect all existing resources that support POST to suddenly return a new header or for all new specifications constraining POST to be aware of its existence and require it, but it is a reasonable requirement for new specifications such as LDP.

5.2.3.14 LDP servers that allow creation of LDP-RSs via POST MUST allow clients to create new members by enclosing a request entity body with a `Content-Type` request header whose value is `application/ld+json` [JSON-LD].

5.2.4 HTTP PUT

Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on creation or update must be advertised to clients.

5.2.4.1 LDP servers SHOULD NOT allow HTTP PUT to update a LDPC's containment triples; if the server receives such a request, it SHOULD respond with a 409 (Conflict) status code.

5.2.4.2 LDP servers that allow LDPR creation via PUT SHOULD NOT re-use URIs.

5.2.5 HTTP DELETE
Per [RFC7231], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

5.2.5.1 When a contained LDPR is deleted, the LDPC server **MUST** also remove the corresponding containment triple, which has the effect of removing the deleted LDPR from the containing LDPC.

Non-normative note: The LDP server might perform additional actions, as described in the normative references like [RFC7231]. For example, the server could remove membership triples referring to the deleted LDPR, perform additional cleanup tasks for resources it knows are no longer referenced or have not been accessed for some period of time, and so on.

5.2.5.2 When a contained LDPR is deleted, and the LDPC server created an associated LDP-RS (see the **LDPC POST** section), the LDPC server **MUST** also delete the associated LDP-RS it created.

5.2.6 HTTP HEAD

Note that certain LDP mechanisms rely on HTTP headers, and HTTP recommends that **HEAD** responses include the same headers as **GET** responses. LDP servers must also include HTTP headers on responses to **OPTIONS**, see section 4.2.8 HTTP OPTIONS. Thus, implementers supporting **HEAD** should also carefully read the section 5.2.2 HTTP GET and section 5.2.8 HTTP OPTIONS.

5.2.7 HTTP PATCH

Per [RFC5789], this HTTP method is optional and this specification does not require LDP servers to support it. When a LDP server supports this method, this specification imposes the following new requirements for LDPCs.

Any server-imposed constraints on LDPR creation or update **must be advertised** to clients.

5.2.7.1 LDP servers are **RECOMMENDED** to support HTTP PATCH as the preferred method for updating a LDPC's minimal-container triples.

5.2.8 HTTP OPTIONS

This specification imposes the following new requirements on HTTP **OPTIONS** for LDPCs. Note that support for this method is required for LDPCs, since it is required for LDPRs and LDPCs adhere to LDP-RS requirements.

5.2.8.1 When responding to requests whose **request-URI** is a **LDP-NR with an associated LDP-RS**, a LDP server **MUST** provide the same HTTP **Link** response header as is **required in the create response**.

5.3 Basic

The following section contains normative clauses for **Linked Data Platform Basic Container**.

5.3.1 General

5.3.1.1 Each LDP Basic Container **MUST** also be a conforming LDP Container in section 5.2 **Container** along with the following restrictions in this section. LDP clients **MAY** infer the following triple: whose subject is the LDP Basic Container, whose predicate is **rdf:type**, and whose object is **ldp:Container**, but there is no requirement to materialize this triple in the LDP-BC representation.

5.4 Direct

The following section contains normative clauses for **Linked Data Platform Direct Container**.

5.4.1 General

5.4.1.1 Each LDP Direct Container **MUST** also be a conforming LDP Container in section 5.2 **Container** along the following restrictions. LDP clients **MAY** infer the following triple: whose subject is the LDP Direct Container, whose predicate is **rdf:type**, and whose object is **ldp:Container**, but there is no requirement to materialize this triple in the LDP-DC representation.

5.4.1.2 LDP Direct Containers **SHOULD** use the **ldp:member** predicate as a LDPC's membership predicate if there is no obvious predicate from an application vocabulary to use. The state of a LDPC includes information about which resources are its members, in the form of membership triples that follow a consistent pattern. The LDPC's state contains enough information for clients to discern the membership predicate, the other consistent membership value used in the container's membership triples (**membership-constant-URI**), and the position (subject or object) where those URIs occurs in the membership triples. Member resources can be any kind of resource identified by a URI, LDPR or otherwise.

5.4.1.3 Each LDP Direct Container representation **MUST** contain exactly one triple whose subject is the LDPC URI, whose predicate is the **ldp:membershipResource**, and whose object is the LDPC's **membership-constant-URI**. Commonly the LDPC's URI is the **membership-constant-URI**, but LDP does not require this.

5.4.1.4 Each LDP Direct Container representation **MUST** contain exactly one triple whose subject is the LDPC URI, and whose predicate is either **ldp:hasMemberRelation** or **ldp:isMemberOfRelation**. The object of the triple is constrained by other sections, such as **ldp:hasMember** or **ldp:isMemberOf**, based on the membership triple pattern used by the container.

5.4.1.4.1 LDP Direct Containers whose membership triple pattern is (**membership-constant-URI**, **membership-predicate**, **member-derived-URI**) **MUST** contain exactly one triple whose subject is the LDPC URI, whose predicate is **ldp:hasMemberRelation**, and whose object is the URI of **membership-predicate**.

5.4.1.4.2 LDP Direct Containers whose membership triple pattern is (**member-derived-URI**, **membership-predicate**, **membership-constant-URI**) **MUST** contain exactly one triple whose subject is the LDPC URI, whose predicate is **ldp:isMemberOfRelation**, and whose object is the URI of **membership-predicate**.
5.4.1.5 LDP Direct Containers MUST behave as if they have a (LDPC URI, ldp:insertedContentRelation, ldp:MemberSubject) triple, but LDP imposes no requirement to materialize such a triple in the LDP-DC representation. The value ldp:MemberSubject means that the member-derived-URI is the URI assigned by the server to a document it creates; for example, if the client POSTs content to a container that causes the container to create a new LDPR, ldp:MemberSubject says that the member-derived-URI is the URI assigned to the newly created LDPR.

5.4.2 HTTP POST

5.4.2.1 When a successful HTTP POST request to a LDPC results in the creation of a LDPR, the LDPC MUST update its membership triples to reflect that addition, and the resulting membership triple MUST be consistent with any LDP-defined predicates it exposes. A LDP Direct Container's membership triples MAY also be modified via through other means.

5.4.3 HTTP DELETE

5.4.3.1 When a LDPR identified by the object of a membership triple which was originally created by the LDP-DC is deleted, the LDPC server MUST also remove the corresponding membership triple.

5.5 Indirect

The following section contains normative clauses for Linked Data Platform Indirect Container.

5.5.1 General

5.5.1.1 Each LDP Indirect Container MUST also be a conforming LDP Direct Container as described in section 5.4 Direct, along with the following restrictions. LDP clients MAY infer the following triple: one whose subject is LDP Indirect Container, whose predicate is rdf:type, and whose object is ldp:Container, but there is no requirement to materialize this triple in the LDP-IC representation.

5.5.1.2 LDP Indirect Containers MUST contain exactly one triple whose subject is the LDPC URI, whose predicate is ldp:insertedContentRelation, and whose object ICR describes how the member-derived-URI in the container's membership triples is chosen. The member-derived-URI is taken from some triple (S, P, O) in the document supplied by the client as input to the create request; if ICRs value is P, then the member-derived-URI is O. LDP does not define the behavior when more than one triple containing the predicate P is present in the client's input. For example, if ICR contains multiple member-derived-URIs, LDP does not define the behavior when more than one triple containing the predicate P is present in the client's input. For example, if the client POSTs RDF content to a container that creates the container to create a new LDPR-DS, and that content contains the triple (S, P, O), member-derived-URI is O. One consequence of this definition is that indirect container member creation is only well-defined by LDP when the document supplied by the client as input to the create request has an RDF media type.

5.5.2 HTTP POST

5.5.2.1 LDPCs whose ldp:insertedContentRelation triple has an object other than ldp:MemberSubject and that create new resources MUST add a triple to the container whose subject is the container's URI, whose predicate is ldp:contains, and whose object is the newly created resource's URI (which will be different from the member-derived-URI in this case). This ldp:contains triple can be the only link from the container to the newly created resource in certain cases.

6. Notable information from normative references

This section is non-normative.

While readers, and especially implementers, of LDP are assumed to understand the information in its normative references, the working group has found that certain points are particularly important to understand. For those thoroughly familiar with the referenced specifications, these points might seem obvious, yet experience has shown that few non-experts find all of them obvious. This section enumerates these topics; it is simply re-stating (non-normatively) information locatable via the normative references.

6.1 Architecture of the World Wide Web

This section is non-normative.

Reference: [WEBARCH]

6.1.1 LDP membership is not exclusive; this means that the same resource (LDPR or not) can be a member of more than one LDPC.

6.1.2 LDP servers should not re-use URIs, regardless of the mechanism by which members are created (POST, PUT, etc.). Certain specific cases exist where a LDPC server might delete a resource and then later re-use the URI when it identifies the same resource, but only when consistent with Web architecture. While it is difficult to provide absolute implementation guarantees of non-reuse in all failure scenarios, re-using URIs creates ambiguities for clients that are best avoided.

6.2 HTTP 1.1

This section is non-normative.

Reference: [RFC7230], [RFC7231], [RFC7232]

6.2.1 LDP servers can support representations beyond those necessary to conform to this specification. These could be other RDF formats, like N3 or N Triples, but non-RDF formats like HTML [HTML401] and JSON [RFC4627] would likely be common. HTTP content negotiation ([RFC7231] Section 3.4 - Content Negotiation) is used to select the format.

6.2.2 LDPRs can be created, updated and deleted using methods not defined in this document, for example through application-specific means, SPARQL UPDATE, etc. [SPAROL-UPDATE], as long as those methods do not conflict with this specification's normative requirements.

6.2.3 LDP servers remove the resource identified by the Request-URI in response to a successful HTTP DELETE request. After such a request, a subsequent HTTP GET on the same Request-URI usually results in a 404 (Not found) or 410 (Gone) status code, although HTTP allows others.

http://www.w3.org/TR/ldp/ 15/20
6.2.4 LDP servers can alter the state of other resources as a result of any HTTP request, especially when non-safe methods are used ([RFC7231] section 4.2.1). For example, it is acceptable for the server to remove triples from other resources whose subject or object is the deleted resource as the result of a successful HTTP DELETE request. It is also acceptable and common for LDP servers to not do this – the server's behavior can vary, so LDP clients cannot depend on it.

6.2.5 LDP servers can implement HTTP PATCH to allow modifications, especially partial replacement, of their resources. No minimal set of patch document formats is mandated by this document or by the definition of PATCH [RFC5789].

6.2.6 When the Content-Type request header is absent from a request, LDP servers might infer the content type by inspecting the entity body contents ([RFC7231] section 3.1.1.5).

6.3 RDF

This section is non-normative.

Reference: [rdf11-concepts]

6.3.1 The state of a LDPR can have triples with any subject(s). The URL used to retrieve the representation of a LDPR need not be the subject of any of its triples.

6.3.2 The representation of a LDPC can include an arbitrary number of additional triples whose subjects are the members of the container, or that are from the representations of the members (if they have RDF representations). This allows an LDP server to provide clients with information about the members without the client having to do a GET on each member individually.

6.3.3 The state of a LDPR can have more than one triple with an rdf:type predicate.

7. HTTP Header Definitions

7.1 The Accept-Post Response Header

NOTE

The LDP Working Group proposes incorporation of the features described in this section.

The Accept-Post header has applicability beyond LDP as outlined in this IETF draft [Accept-Post].

This specification introduces a new HTTP response header Accept-Post used to specify the document formats accepted by the server on HTTP POST requests. It is modelled after the Accept-Patch header defined in [RFC5789].

7.1.1 The syntax for Accept-Post, using the ABNF syntax defined in Section 1.2 of [RFC7231], is:

Accept-Post = "Accept-Post" :: " : " # media-range

The Accept-Post header specifies a comma-separated list of media ranges (with optional parameters) as defined by [RFC7231]. Section 5.3.2. The Accept-Post header, in effect, uses the same syntax as the HTTP Accept header minus the optional accept-params BNF production, since the latter does not apply to Accept-Post.

7.1.2 The Accept-Post HTTP header SHOULD appear in the OPTIONS response for any resource that supports the use of the POST method. The presence of the Accept-Post header in response to any method is an implicit indication that POST is allowed on the resource identified by the Request-URI. The presence of a specific document format in this header indicates that that specific format is allowed on POST requests to the resource identified by the Request-URI.

7.1.3 IANA Registration Template

The Accept-Post response header must be added to the permanent registry (see [RFC3864]).

Header field name: Accept-Post
Applicable Protocol: HTTP
Author/Change controller: W3C
Specification document: this specification

7.2 Preferences on the Prefer Request Header

7.2.1 Summary

This specification introduces new parameters on the HTTP Prefer request header's return=representation preference [RFC7240], used optionally by clients to supply a hint to help the server form a response that is most appropriate to the client's needs. The LDP-defined parameters suggest the portion(s) of a resource's state that the client application is interested in and, if received, is likely to be processed. LDP Containers with large numbers of associated documents and/or members will have large representations, and many client applications may be interested in processing only a subset of the LDPC's information (for example, only membership triples or only containment triples), resulting in a potentially large savings in server, client, and network processing.

Non-normative note: LDP server implementers should carefully consider the effects of these preferences on caching, as described in section 2 of [RFC7240].

Non-normative note: [RFC7240] recommends that server implementers include a Preference-Applied response header when the client cannot otherwise determine the server's behavior with respect to honoring hints from the response content. Examples illustrate some cases where the header is unnecessary.

http://www.w3.org/TR/ldp/
7.2.2 Specification

7.2.2.1 The include hint defines a subset of a LDPR's content that a client would like included in a representation. The syntax for the include parameter of the HTTP Prefer request header's return=representation preference [RFC7240] is:

```
include-parameter = "include" *WSP "=" *WSP ldp-uri-list
```

Where WSP is whitespace [RFC5234], and ldp-uri-list is a double-quoted blank-delimited unordered set of URIs whose ABNF is given below. The generic preference BNF [RFC7240] allows either a quoted string or a token as the value of a preference parameter; LDP assigns a meaning to the value only when it is a quoted string of the form:

```
ldp-uri-list = DQUOTE *WSP URI *[ 1*WSP URI ] *WSP DQUOTE
```

where DQUOTE is a double quote [RFC5234], and URI is an absolute URI with an optional fragment component [RFC3986].

7.2.2.2 The omit hint defines a subset of a LDPR's content that a client would like omitted from a representation. The syntax for the omit parameter of the HTTP Prefer request header's return=representation preference [RFC7240] is:

```
omit-parameter = "omit" *WSP "=" *WSP ldp-uri-list
```

Where WSP and ldp-uri-list are defined as above for include.

7.2.2.3 When LDP servers receive a request with conflicting hints, this specification imposes no requirements on their behavior. They are free to reject the request, process it applying some subset of the hints, or anything else appropriate to the server. [RFC7240] suggests treating similar requests as though none of the conflicting preferences were specified.

7.2.2.4 This specification defines the following URIs for clients to use with include and omit parameters. It assigns no meaning to other URIs, although other specifications MAY do so.

<table>
<thead>
<tr>
<th>Category</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containment triples</td>
<td><a href="http://www.w3.org/ns/ldp#PreferContainment">http://www.w3.org/ns/ldp#PreferContainment</a></td>
</tr>
<tr>
<td>Membership triples</td>
<td><a href="http://www.w3.org/ns/ldp#PreferMembership">http://www.w3.org/ns/ldp#PreferMembership</a></td>
</tr>
<tr>
<td>Minimal-container triples</td>
<td><a href="http://www.w3.org/ns/ldp#PreferMinimalContainer">http://www.w3.org/ns/ldp#PreferMinimalContainer</a></td>
</tr>
<tr>
<td>or the equivalent but deprecated term</td>
<td><a href="http://www.w3.org/ns/ldp#PreferEmptyContainer">http://www.w3.org/ns/ldp#PreferEmptyContainer</a></td>
</tr>
</tbody>
</table>

Non-normative note: all currently defined URIs are only coherent for LDP-RSs, and in fact only for LDPCs, however in the future it is possible that additional URIs with other scopes of applicability could be defined.

7.2.3 Examples

This section is non-normative.

If we assume a container like the one below:

```
EXAMPLE 19
# The following is the representation of
# http://example.org/netWorth/nw1/assets/
@base <http://example.org/netWorth/nw1/assets/>.  
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.  
@prefix o: <http://example.org/ontology#>.
<>  
  a ldp:DirectContainer;  
  dcterms:title "The assets of JohnZSmith";  
  ldp:membershipResource <http://example.org/netWorth/nw1/>;  
  ldp:hasMemberRelation o:asset;  
  ldp:insertedContentRelation ldp:MemberSubject.  
<http://example.org/netWorth/nw1/>  
  a o:NetWorth;  
  o:asset <a1>, <a3>, <a2>.  
<a1>  
  a o:Stock;  
  o:marketValue 100.00 .  
<a2>  
  a o:Cash;  
  o:marketValue 50.00 .  
<a3>  
  a o:RealEstateHolding;  
  o:marketValue 300000 .
```

Clients interested only in information about the container (for example, which membership predicate it uses) might use this hint on a GET request: Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMinimalContainer"

A server that honors this hint would return a following response containing the HTTP header Preference-Applied: return=representation and this representation:

```
EXAMPLE 20
GET /netWorth/nw1/assets/ HTTP/1.1  
Host: example.org  
Accept: text/turtle  
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

http://www.w3.org/TR/ldp/
Clients interested only in information about the container (same as before) might use this hint instead: 

```
Prefer: return=representation; omit="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferContainment"
```

Note: Treating the `omit` parameter value is equivalent to the preceding `include` parameter value, they may not be equivalent in the future due to the definition of minimal-container triples. Clients should preferentially use the `include` parameter, as it more precisely communicates their needs.

A LDP 1.0 server that honors this hint would return the following response. Servers implementing later versions of LDP might return substantively different responses.

Request to `http://example.org/netWorth/nw1/assets/`:

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; omit="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferContainment"
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type",
     <http://www.w3.org/ns/ldp#Resource>; rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Transfer-Encoding: chunked

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
@prefix o: <http://example.org/ontology#>.

<http://example.org/netWorth/nw1/assets/>
 a ldp:DirectContainer;
 dcterms:title "The assets of JohnZSmith";
 ldp:membershipResource <http://example.org/netWorth/nw1/>;
 ldp:hasMemberRelation o:asset;
 ldp:insertedContentRelation ldp:MemberSubject.
```

Clients interested only in information about the container (for example, which membership predicate it uses) and its membership might use this hint on a `GET` request:

```
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

A server that honors this hint would return (at least) the following response, and perhaps only this (it might well omit containment triples if they are not specifically requested). In cases like this example, where a client can detect from the content that its hints were honored (the presence of the predicates `dcterms:title` and `o:asset` demonstrate this in the representation below), there is no need for the server to include a `Preference-Applied` response header in many common cases like a `200 (OK)` response. In other cases, like status code `303`, the header would still be required for the client to know that the `303` response entity is a representation of the resource identified by the `Location` URI instead of a short hypertext note (one with a hyperlink to the same URI reference provided in the `Location` header field [RFC7231]).

Request to `http://example.org/netWorth/nw1/assets/`:

```
GET /netWorth/nw1/assets/ HTTP/1.1
Host: example.org
Accept: text/turtle
Prefer: return=representation; include="http://www.w3.org/ns/ldp#PreferMembership http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

Response:

```
HTTP/1.1 200 OK
```

http://www.w3.org/TR/ldp/
8. Link Relations

The intent is that these link relations will be registered with IANA per [RFC5988] section 6.2.1.

8.1 describedby

The contents of this section were originally taken from [POWDER] appendix D, and then modified to comply with the current registration template. The pre-LDP IANA link relation registry entry for describedby refers to a different section of [POWDER] that was substantively updated in an erratum, and that section was not actually the normative definition of the link relation. Since we expect no update to [POWDER] that incorporates the erratum or fixes the registry link, this superseding registration approach is being taken.

The following Link Relationship will be submitted to IANA for review, approval, and inclusion in the IANA Link Relations registry.

Relation Name: describedby
Description: The relationship A describedby B asserts that resource B provides a description of resource A. There are no constraints on the format or representation of either A or B, neither are there any further constraints on either resource.
Notes: Descriptions of resources may be socially sensitive, may require processing to be understood and may or may not be accurate. Consumers of descriptive resources should be aware of the source and chain of custody of the data. Security considerations for URIs (Section 7 of RFC 3986) and IRIs (Section 8 of RFC 3987) apply to the extent that describing resources may affect consumers' decisions about how or whether to retrieve those resources.

9. Security Considerations

This section is non-normative.

As with any protocol that is implemented leveraging HTTP, implementations should take advantage of the many security-related facilities associated with it and are not required to carry out LDP operations that may be in contradistinction to a particular security policy in place. For example, when faced with an unauthenticated request to replace system critical RDF statements in a graph through the PUT method, applications may consider responding with the 401 status code (Unauthorized), indicating that the appropriate authorization is required. In cases where the provided authentication fails to meet the requirements of a particular access control policy, the 403 status code (Forbidden) can be sent back to the client to indicate this failure to meet the access control policy.

A. Acknowledgements

This section is non-normative.

The following people have been instrumental in providing thoughts, feedback, reviews, content, criticism and input in the creation of this specification:


B. Change History

This section is non-normative.

The change history is up to the editors to insert a brief summary of changes, ordered by most recent changes first and with heading from which public draft it has been changed from.

Summary

Summary of notable changes from the Proposed Recommendation.

- Minor rewording about provided authentication in Security Considerations
- Added sentence before example 3 in the introduction to Linked Data Platform Containers

C. References
C.1 Normative references

[DC-TERMS]  

[JSON-LD]  
Manu Sporny; Gregg Kellogg; Markus Lanthaler. JSON-LD 1.0, 16 January 2014. W3C Recommendation. URL: http://www.w3.org/TR/2014/REC-json-ld-20140116.

[RFC2119]  

[RFC3664]  

[RFC3986]  

[RFC3987]  

[RFC5023]  

[RFC5234]  

[RFC5789]  

[RFC5988]  

[RFC6585]  

[RFC7230]  

[RFC7231]  

[RFC7232]  

[RFC7240]  

[WEBARCH]  

[rdf-schema]  

[rdf11-concepts]  
Richard Cyganiak; David Wood; Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, 25 February 2014. W3C Recommendation. URL: http://www.w3.org/TR/rdf11-concepts/.

[turtle]  

C.2 Informative references

[Accept-Post]  

[HTML401]  

[LDP-PAGING]  
S. Speicher; J. Arwe; A. Malikotra. Linked Data Platform Paging, Candidate Recommendation. URL: http://www.w3.org/TR/ldp-paging/.

[LDP-tests]  

[LDP-UCR]  
Steve Battle; Steve Speicher. Linked Data Platform Use Cases and Requirements, 13 March 2014. W3C Note. URL: http://www.w3.org/TR/ldp-ucr/.

[LINKED-DATA]  

[POWDER]  

[RFC4627]  

[SPARQL-UPDATE]  
Paul Geaaron; Alexandre Passant; Axel Polleres. SPARQL 1.1 Update, 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-update/.

[sparql11-query]  
Steven Harris; Andy Seaborne. SPARQL 1.1 Query Language, 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-query/.
Abstract

This primer provides an introduction to the Linked Data Platform (LDP), with examples illustrating the principal concepts such as the notion of an LDP resource, and the LDP container and how they can be used by Web clients. Two sample scenarios show how an LDP client can interact with a LDP server in the context of a read-write Linked Data application i.e. how to use HTTP for accessing, updating, creating and deleting resources from servers that expose their resources as Linked Data.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This is the first draft of LDP Primer Note from the W3C LDP working group.

This document was published by the Linked Data Platform Working Group as an Editor’s Draft. If you wish to make comments regarding this document, please send them to public-ldp-comments@w3.org (subscribe, archives). All comments are welcome.
Publication as an Editor's Draft does not imply endorsement by the W3C Membership. This is a draft
document and may be updated, replaced or obsoleted by other documents at any time. It is
improper to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy.
W3C maintains a public list of any patent disclosures made in connection with the deliverables of the
group; that page also includes instructions for disclosing a patent. An individual who has actual
knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the in-
formation in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 March 2017 W3C Process Document.

Table of Contents

1. Introduction

2. Online document store example (LDP Basics)
   2.1 Looking up a basic container (GET on an LDP-BC)
   2.2 Discovering the affordances (OPTIONS on an LDP-BC)
   2.3 Creating Containers and Structural Hierarchy
   2.4 Creating an RDF resource (POST an RDF resource to an LDP-BC)
   2.5 Creating a non-RDF (binary) resource (POST an image to an LDP-BC)
   2.6 Update a RDF LDP resource (PUT on an LDP-RS)
   2.7 Deleting a resource (DELETE on an LDPR)

3. Bug Tracker Example (LDP Direct containers)
   3.1 Navigation and Retrieval (GET on an LDP-DC)
   3.2 Creation (POST a resource to an LDP-DC)
   3.3 Deletion (DELETE on an LDPR associated with an LDP-DC)

4. Extended Bug Tracker Example (LDP Indirect containers)
   4.1 Navigation and Retrieval (GET on an LDP-IC)
   4.2 Creation (POST a resource to an LDP-IC)

5. Security

6. LDP Implementations

7. What To Read Next
1. Introduction

The term "Linked Data" [LINKED-DATA] refers to an approach to publishing data that puts linking at the heart of the notion of data, and uses the linking technologies provided by the Web to enable the weaving of a global distributed database. By naming real world entities - be they web resources, physical objects such as the Eiffel Tower, or even more abstract things such as relations or concepts - with http(s) URLs, whose meaning can be determined by dereferencing the document at that URL, and by using the relational framework provided by RDF, data can be published and linked in the same way web pages can. The Linked Data Protocol specifies how web applications can, using the HTTP protocol, find resources and follow links, publish new resources, edit and delete existing ones.

The Primer aims to provide introductory examples and guidance in the use of the LDP protocol, in accordance with best practices [LDP-BP]. For a systematic description of the protocol the reader should consult the normative LDP reference [LDP]. For an overview of the use cases for LDP and the elicited requirements that guided its design, the reader should consult the LDP Use Cases and Requirements [LDP-UCR]; for best practices and guidelines, the reader should consult the LDP Best Practices and Guidelines document [LDP-BP].

Conventions Used in This Document

The examples in this guide are given as a serialization of RDF graphs using the Turtle [turtle] and JSON-LD [json-ld] syntaxes of RDF.

The buttons below can be used to show or hide the available syntaxes.

LDP concepts in a glance

A server hosting Linked Data Platform Resources (LDPRs) may manage two kinds of LDPRs: those resources whose state is represented using RDF (called LDP RDF Sources (LDP-RSs)), and those us-
ing other formats (called LDP Non-RDF Sources (LDP-NRs)) such as HTML files, images, other binary files, etc. Resources respond to retrieval operations using HTTP GET. Often a description conveyed in the response document will describe a specific domain entity; Status, Friendship, Product, Order, Bug, etc. On the other hand, it might contain a description of a number of different concepts. The links contained in the descriptions lead to the subsequent discovery and processing of other resources. Affordances offered by the server make discoverable the forward paths in the application. Together the resources, links and associated affordances together specify what might be termed the API.

The LDP protocol covers read and write interactions with Resources. Writable aspects include creation of new resources (using POST or PUT), updates (using PUT or PATCH), and deletion of resources. Resource creation is an essential feature providing structured creation of resources. Affordances published by the server show that some Resources can be used to create other Resources. This common pattern is often seen in cases where one resource is made up of a number of others, e.g. a Document Store consists of Documents, a Bug Tracker consists of Bug Reports, a Photo Album consists of Photos, the Net Worth of a person consists of Assets and Liabilities. LDP defines creation for a special kind of Resource called a Container (LDPC), which is able to respond to requests to create new resources, in addition to the general mechanisms HTTP defines. During creation the created resource is added to its Container and a containment link between the Container and the new entry is made.

Therefore a LDPC is a specialization of a LDP-RS representing a collection of links to LDPRs or information resources [[WEBARCH]] that responds to client requests for creation, modification, and/or enumeration of its linked members and documents. The simplest container is the Basic Container (LDP-BC). It defines the basic containment described using a generic vocabulary. This can be used in a generic storage service to manage a containment hierarchy of arbitrary resources.

---

Fig. 1 Generic document storage using a Basic Container.
Such servers do not impose any restriction on LDPRs and generally act as storage systems without any domain specific application logic and vocabularies. The first scenario in this document concerns a document storage system based on Basic Containers.

A Direct Container is a specialisation of a Basic Container. Additional assertions called membership triples which use a domain-specific vocabulary are made by a Direct Container as part of the creation process. The membership triples augment the containment triples maintained by all containers. For example, one aspect of a Product inventory system concerns the how a Direct Container is used for the management of a Product portfolio, where use of existing vocabulary is preferable.

![Fig. 2 Using domain vocabulary with a Direct container.](image1)

Direct Container membership triples can be about subjects other than the Container resource. An example is a Photo management application where a Photo Container is used for the management of Photos, and where membership triples then express the relationship between a User and a Photo.

![Fig. 3 Membership triples with a non-Container subject.](image2)

Another common pattern is where different facets of a Resource be managed using multiple Containers. For example, a Bug Report has an associated list of Comments as well as supportive media resources.
One important usage concerns using LDP to expose the data and services of existing applications. These systems impose restrictions on LDPRs since the LDP interactions should consider the constraints of the underlying business logic and data model. The bug tracker example presented in the latter part of this primer is an example of an application specific LDP server.

NOTE

Formal definitions of the terms LDPR, LDPC, and other concepts introduced by LDP can be found in the 'Terminology' section of the Linked Data Platform 1.0 specification [LDP]

The following provide a set of examples to show the Linked Data Platform interactions. Note, this is a primer and should not be considered as a canonical example of ideal LDP modeling.

2. Online document store example (LDP Basics)

This section provides a set of examples of using an online document store application. These examples will demonstrate the behaviour of both types of LDPRs and LDP Basic Containers. Registration with the online document store application by a user results in some data storage space (a root Basic Container) where the application's web resources are stored. Using this root Basic Container a user can create new documents and also child containers to further organize her documents stored in this application.

APIs of web applications are commonly documented by listing valid operations which can operate on URLs, where the URLs are described as templates. A description of our exemplary LDP based docu-
ment store is in the following table. We note with emphasis that it is important for servers to use links as the main mechanism to reveal the location of resources. If it would be necessary to encode templates into client applications, this would be a strong indicator that the design breaches a number of good design principles.

<table>
<thead>
<tr>
<th>Path</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/{username}/</td>
<td>GET</td>
<td>Lists all the documents in the root container.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Create a new document under the root container.</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>Update the description and/or list of files of the root container.</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>Update the description and/or list of files of the root container.</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Not allowed.</td>
</tr>
<tr>
<td>/{username}/{{document}/}/*</td>
<td>GET</td>
<td>Retrieve the document.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Discovered from the resource affordances.</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>Update the document.</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>Partial update to the document if PATCH is supported.</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Delete the document.</td>
</tr>
<tr>
<td>/<em>/</em></td>
<td>OPTIONS</td>
<td>Discover the allowed operations over a resource</td>
</tr>
<tr>
<td></td>
<td>HEAD</td>
<td>Only retrieve meta-information about a resource</td>
</tr>
</tbody>
</table>

In this example, we will see how Alice, a user of this system, does read / write management of documents using the LDP protocol. A typical interaction with the system would start with Alice registering as a user. It is likely that registration would be a LDP based interaction, but this aspect is out of scope of this example. A consequence of the registration is allocation of space for the storage of documents, and communication of this URL to the user, e.g. a basic container at http://example.org/alice/. This section describes a typical flow of interactions where Alice firsts reads the root document and discovers its affordances. This is followed by subsequent examples of creation, update and delete, and finishes with how the client is able to create nested structure from the containers.
2.1 Looking up a basic container (GET on an LDP-BC)

First Alice looks up her storage by retrieving the LDP Basic Container assigned to her to hold her documents. Alice's LDP client does this by doing a GET request on the URI, http://example.org/alice/.

EXAMPLE 1: Request - basic container retrieval

Turtle:GET /alice/ HTTP/1.1
Host: example.org
Accept: text/turtle

JSON-LD:GET /alice/ HTTP/1.1
Host: example.org
Accept: application/ld+json

As her document storage was just created, it is an empty container.
As shown in the example, in addition to the RDF representation of the Basic Container using the requested media type the server provides an etag of the resource representation and Link headers advertising that the requested resource is indeed an LDP Basic Container and it will support the LDP interaction model.

In addition, the response also contains "Allow", "Accept-Post", "Accept-Patch" headers. The "Allow" header advertises which HTTP operations are supported by this LDP Basic Container resource. In this
example, it supports OPTIONS, HEAD, GET, POST, PUT, and PATCH HTTP verbs. The "Accept-Post" and "Accept-Patch" headers advertise which are the media types supported by the POST and PATCH method respectively.

NOTE

The Linked Data Platform 1.0 specification [LDP] says that all LDP servers must support the Turtle media type for LDP-RS resources and should support JSON-LD media type.

2.2 Discovering the affordances (OPTIONS on an LDP-BC)

In the previous example, we saw that Alice can discover what operations are allowed on a resource by doing a GET request on the resource. As an alternative, she can use the OPTIONS operation to learn of the permitted operations on any given resource.

EXAMPLE 3: Request - retrieving OPTIONS of a basic container

OPTIONS /alice/ HTTP/1.1
Host: example.org

EXAMPLE 4: Response - retrieving OPTIONS of a basic container

HTTP/1.1 204 No Content
Allow: OPTIONS, HEAD, GET, POST, PUT, PATCH
Accept-Post: text/turtle, application/ld+json, image/bmp, image/jpeg
Accept-Patch: text/ldpatch
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type", <http://www.w3.org/ns/ldp#Resource>; rel="type"

According to the response, HTTP operations {OPTIONS, HEAD, GET, POST, PUT, PATCH} are allowed on her root container. In addition to the allowed operations, Accept-Post and Accept-Patch provides which media types are supported by the respective operations. The rel="type" Link header advertises that this resource supports LDP protocol and it is an LDP Basic Container.

In this case, the response tells Alice's LDP client that this is an LDP-Basic Container and the container allows her to POST things of both RDF types (text/turtle, application/ld+json) and images (image/bmp and image/jpeg).
2.3 Creating Containers and Structural Hierarchy

In order for the client to introduce hierarchy to the management of documents, the document store allows creation of container resources, enabling Alice to create a container hierarchy to organise her documents. This can be done by POSTing (a child) container representation to a (parent) container. For instance, it enables Alice to create a child container which she intends to use for image storage.

![Diagram of Document Container and Document Resource](image)

*Fig. 5 Child Containers inside a Basic Container.*

**EXAMPLE 5: State of Alice's document store before creating the photo (child) container**

Turtle:

```
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://example.org/alice/> a ldp:Container, ldp:BasicContainer ;
    dcterms:title "Alice’s data storage on the Web" ;
```

JSON-LD:

```
{
    "@context": {
        "dcterms": "http://purl.org/dc/terms/",
        "ldp": "http://www.w3.org/ns/ldp#" },
    "@id": "http://example.org/alice/",
    "@type": [ "ldp:Container", "ldp:BasicContainer" ],
    "dcterms:title": "Alice’s data storage on the Web",
    "ldp:contains": [ "@id": "http://example.org/alice/foaf" ]
}
```

To create a new container for managing photos, Alice POSTs a representation of a container (LDP-BC) to the root container. Alice expresses her intention that the newly created resource should be an LDP Basic Container by including a Link header in the request with the relationship "type".
If the POST is successful, the server responds with the location of the newly created container for the photos.

EXAMPLE 7: Response - creating a new container

HTTP/1.1 201 Created
Location: http://example.org/alice/photos/
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
EXAMPLE 8: State of Alice's document store after creating the photo (child) container

Turtle: @prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://example.org/alice/> a ldp:Container, ldp:BasicContainer ;
   dcterms:title "Alice’s data storage on the Web";

JSON-LD:
{
   "@context": "https://dvcs.w3.org/hg/ldpwg/raw-file/default/ldp-primer/context.json",
   "@id": "http://example.org/alice/",
   "@type": [ldp:Container, ldp:BasicContainer],
   "dcterms:title": "Alice’s data storage on the Web",
   "ldp:contains": [
       { "@id": "http://example.org/alice/foaf" },
       { "@id": "http://example.org/alice/photos/" }
   ]
}

and the photo container will look like the following.
2.4 Creating an RDF resource (POST an RDF resource to an LDP-BC)

Alice can upload a social profile document to her store, by POSTing her FOAF personal profile document to her LDP-BC at the root of her document store. Note, the Slug header offers a hint to the server about the URL of the resource to be created. Alice also indicates that the newly created resource should be an LDP Resource by including a Link header in the request with the relationship "type".

The FOAF document includes statements about the resource to be created and other resources relative to the resource to be created. According the LDP specification, Alice can use an empty relative URI (<> in the request entity body to refer to the resource to be created.

EXAMPLE 9: State of Alice's newly created photo container

Turtle:@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://example.org/alice/photos/> a ldp:Container, ldp:BasicContainer;
  dcterms:title "Photos of Alice" ;
  dcterms:description "This container will contain photos of Alice." .

JSON-LD:
  "@context": "https://dvcs.w3.org/hg/ldpwg/raw-file/default/ldp-primer/context.json",
  "@id": "/app/product2/",
  "@type": [ "ldp:Container", "ldp:BasicContainer" ],
  "dcterms:title": "Photos of Alice",
  "dcterms:description": "This container will contain photos of Alice." }
EXAMPLE 10: Request - creating an RDF resource

Turtle: POST /alice/ HTTP/1.1
Host: example.org
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Slug: foaf
Content-Type: text/turtle

@prefix dc: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<> a foaf:PersonalProfileDocument;
    foaf:primaryTopic <#me> ;
    dc:title 'Alice’s FOAF file' .

<#me> a foaf:Person;
    foaf:name 'Alice Smith' .

JSON-LD: POST /alice/ HTTP/1.1
Host: example.org
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Slug: foaf
Content-Type: application/ld+json

{
    "@context": {
        "dcterms": "http://purl.org/dc/terms/",
        "foaf": "http://xmlns.com/foaf/0.1/" },
    "@id": "",
    "@type": "foaf:PersonalProfileDocument",
    "foaf:primaryTopic": {
        "@id": "#me",
        "@type": "foaf:Person",
        "foaf:name": "Alice Smith"
    },
    "dcterms:title": "Alice’s FOAF file"
}
The response to the create request provides a Link to the newly created resource using the Location header. In this case, the server has honored the hint provided by the slug header and created the new resource in the URL http://example.org/alice/foaf.

Knowing the URL of the newly created resource, Alice can check the container again to confirm that the container correctly contains the newly created resource.

EXAMPLE 11: Response - creating an RDF resource

HTTP/1.1 201 Created
Location: http://example.org/alice/foaf
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0

EXAMPLE 12: Request - basic container retrieval after resource created

GET /alice/ HTTP/1.1
Host: example.org
Accept: text/turtle, application/ld+json
2.5 Creating a non-RDF (binary) resource (POST an image to an LDP-BC)

Next, Alice wants to upload a photo of herself to the document storage. She can create an image by POSTing it in the same way she created the RDF document.
The outcome of creating a non-RDF is similar to creating a RDF resource. If successful, the server will return a 201 success code with a Location header that points to the created resource. Furthermore, in the case of binary resources the server may create an additional file to maintain the metadata about the binary file. In the above example, the server creates a new LDP-RS to maintain metadata about the binary resource such as creation date, owner, etc. and this metadata resource is advertised using a Link header with the relation "describedby".

Similar to creating a RDF resource (LDP-RS), a containment triple will be added to the container when a non-RDF (LDP-NR) is created. Thus, the representation of the LDP container after creating the image looks like the following.

---

**EXAMPLE 14: Request - creating a non-RDF resource**

```
POST /alice/ HTTP/1.1
Host: example.org
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Slug: avatar
Content-Type: image/png
Content-Length: 1020

### binary data ###
```

**EXAMPLE 15: Response - creating a non-RDF resource**

```
HTTP/1.1 201 Created
Location: http://example.org/alice/avatar
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Link: <http://example.org/alice/avatar/meta>; rel="describedby"
Content-Length: 0
```
2.6 Update a RDF LDP resource (PUT on an LDP-RS)

After creating the image as shown in the previous example, Alice now wants to update her FOAF profile with a link to the image. After retrieving her FOAF profile using a HTTP GET operation, she uses HTTP PUT to update the document by amending the RDF with a link to her photo.

In this example, Alice's LDP client sends the E-tag of the resource representation that it retrieved previously to prevent any lost update problems.
EXAMPLE 17: Request - updating a RDF resource

Turtle:
PUT /alice/foaf HTTP/1.1
Host: example.org
If-Match: W/"123454321"
Content-Type: text/turtle

@prefix dc: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<> a foaf:PersonalProfileDocument;
   foaf:primaryTopic <#me> ;
   dc:title "Alice’s FOAF file" .

<#me> a foaf:Person;
   foaf:name "Alice Smith" ;
   foaf:img <http://example.org/alice/avatar> .

JSON-LD:

   "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "foaf": "http://xmlns.com/foaf/0.1/" },
   "@id": "",
   "@type": "foaf:PersonalProfileDocument",
   "foaf:primaryTopic": {
      "@id":="#me",
      "@type": "foaf:Person",
      "foaf:name": "Alice Smith",
      "foaf:img": { "@id": "http://example.org/alice/avatar" }
   },
   "dcterms:title": "Alice’s FOAF file"

EXAMPLE 18: Response - updating a RDF resource

HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"

If the operation is successful as shown above, the document will be updated with new information.
2.7 Deleting a resource (DELETE on an LDPR)

If Alice decides to delete the image, she can do that with a delete operation.

EXAMPLE 19: Request - deleting a RDF resource

DELETE /alice/avatar HTTP/1.1
Host: example.org

EXAMPLE 20: Response - deleting a RDF resource

HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"

As well as deleting the resource, the server removes the containment triple from the container. For example, a subsequent GET request on the container will return a graph isomorphic to the one shown in the following representation:

NOTE

Alice can also use the PATCH operation to update the resource.
For any subsequent request on the deleted resource, the server will respond with the appropriate HTTP response code.

EXAMPLE 21: Container representation after resource deletion

Turtle:

@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.

<http://example.org/alice/> a ldp:Container, ldp:BasicContainer;
   dcterms:title 'Alice's data storage on the Web' ;

JSON-LD:

{  
   "@context": { 
      "dcterms": "http://purl.org/dc/terms/",
      "ldp": "http://www.w3.org/ns/ldp#" 
   },
   "@id": "http://example.org/alice/",
   "@type": [ "ldp:Container", "ldp:BasicContainer" ],
   "dcterms:title": "Alice’s data storage on the Web",
   "ldp:contains": { "@id": "http://example.org/alice/foaf" }
}

EXAMPLE 22: Request - after deletion

GET /alice/avatar HTTP/1.1
Host: example.org
Accept: image/png

EXAMPLE 23: Response - after deletion

HTTP/1.1 410 Gone

3. Bug Tracker Example (LDP Direct containers)

The previous section provided practical examples of basic LDP interactions using LDP Basic Containers. One of the limitations of LDP Basic Containers is that a fixed LDP vocabulary is used to assert the
relations between a container and its contained resources. However, some scenarios require domain specific vocabulary to be used to list the members of a container. For example, an application which already used Linked Data and its own vocabulary may like to continue using the same vocabulary when migrating to LDP protocol. LDP Direct containers introduce the concept of membership triples allowing the flexibility to define the form of the membership. One of these flexibility points is the ability to select the predicate of the membership triple which can be from a domain-specific vocabulary. This is done using the ldp:hasMemberRelation or ldp:isMemberOfRelation predicate of the Direct Containers.

In addition, in some scenarios it is necessary to add relationships between the newly created resource and some other resource (which is not necessarily the container or another document / information resource). LDP Direct Container allow to define relationships between the container and any other information resource or non-information resource (real world thing) by defining the membership constant subject or the object URI of the membership triples using ldp:membershipResource predicate of the Direct Container. The usage of the ldp:hasMemberRelation predicate as well as the ldp:membershipResource will be explained in the following examples.

NOTE

For more information on information resources (documents) vs real world entities (things) separation please refer to Web Arch (Section 2.2. URI/Resource Relationships), Cool URIs (Section 3. URIs for Real-World Objects), URLs in Data (Section 3. Landing Pages and Records).

The examples in this section will revolve around a very simple Bug Tracker application. Bug Tracker application records the bugs of several products allowing reporting, updating and deleting bugs and products. In contrast to the online document store example, the bug tracker wants to use an existing domain vocabulary, e.g. has_bug, to express membership relationships in the containers. LDP provides the additional interaction capability in the protocol to add the domain specific triples based on the properties defined in the LDP Direct Container.

A RESTful API for a simple Bug Tracker system might be described as follows.

<table>
<thead>
<tr>
<th>Path</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tracker/{product-id}/</td>
<td>GET</td>
<td>Lists the product description and bug reports associated with a product.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Create a new bug report associated with a product.</td>
</tr>
<tr>
<td>Path</td>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>Update the project description.</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>Not supported.</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Delete the project description and associated bug reports.</td>
</tr>
<tr>
<td>/tracker/{product-id}/{bug-id}</td>
<td>GET</td>
<td>Gets the bug report.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Not supported.</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>Update the bug report.</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>Not supported.</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>Delete the bug report.</td>
</tr>
<tr>
<td>/tracker/<em>/</em></td>
<td>OPTIONS</td>
<td>Discover the allowed operations over a resource</td>
</tr>
<tr>
<td></td>
<td>HEAD</td>
<td>Only retrieve meta information about a resource</td>
</tr>
</tbody>
</table>

In the examples in this section, we will only focus on the container representation, creation and deletion of resources because that is where the Basic Containers, Direct Containers, and Indirect Containers differ. Other operations such as updating a resource would be similar to what was illustrated in the previous example.

3.1 Navigation and Retrieval (GET on an LDP-DC)

One of the main use cases of the example bug tracker is to list a given product's bugs. Assuming that a user got a URL of a product by out of band means, she can look it up to get more information including the bugs associated with it. To get the description of the product, a user (or her LDP client) can do a GET request on the URI of the known product resource. LDPR servers must provide text/turtle representations of the requested LDPRs and may provide other RDF format representations such as JSON-LD or RDF/XML using standard HTTP content negotiation.
If the product resource is available, the server responds with the RDF representation of the Direct Container that corresponds to the given product using the requested media type, `text/turtle` or `application/ld+json` in this case.
EXAMPLE 25: Response - Product Lookup
Turtle:HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type", <http://www.w3.org/ns/ldp#DirectContainer>; rel="parent"
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Accept-Patch: text/ldpatch
Content-Length: 246
ETag: W/"123456789"

@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .

<> a ldp:DirectContainer;
  ldp:membershipResource <#it>;
  ldp:hasMemberRelation bt:hasBug;
  dcterms:title "Product description of the LDP Demo product which is also an LDP-DC";

<#it> a bt:Product;
  dcterms:title "LDP Demo";
  bt:hasBug <bug3>, <bug4> .

JSON-LD:HTTP/1.1 200 OK
Content-Type: application/ld+json; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#DirectContainer>; rel="type", <http://www.w3.org/ns/ldp#DirectContainer>; rel="parent"
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Accept-Patch: text/ldpatch
Content-Length: 315
ETag: W/"123456789"

[ 
  { "@context": {
    "dcterms": "http://purl.org/dc/terms/",
    "ldp": "http://www.w3.org/ns/ldp#",
    "bt": "http://example.org/vocab/bugtracker#" },
  "@id": "",
  "@type": [ "ldp:DirectContainer", "bt:Product"],
  "dcterms:title": "Product description of the LDP Demo product which is also an LDP-DC",
  "ldp:contains": [{ "@id": "bug3"}, { "@id": "bug4"}],
  "ldp:hasMemberRelation": { "@id": "bt:hasBug"},
  "ldp:membershipResource": { "@id": "#it"}
},
{ }
The representation of the product contains both information about the product such as the title and the bugs associated with the product and information about the product description such as the title of the product description and other properties of the LDP Container.

As you can see from the Link Header that is returned and the RDF representation of the container, this example uses an LDP Direct Container. LDP-DCs contain both containment triples and membership triples and provide the flexibility to the applications to use domain specific vocabulary in the membership triples. For example, in the above example the LDP-DC manages the member relationship, (<?productURI>, bt:hasBug, <?bugURI>), using the application-specific vocabulary term, bt:hasBug. This is done by defining the ldp:hasMemberRelation predicate of the Direct Container to bt:hasBug (<?directContainerURI>, ldp:hasMemberRelation, bt:hasbug).

NOTE

The Direct Container shown in the above example has the membership triple pattern (membership-constant-URI, membership-predicate, member-derived-URI) using ldp:hasMemberRelation where the constant membership resource is in the subject of the triple and the newly created resources will be added as the object of the triple. It is also possible for the Direct Container to have the membership triple pattern (member-derived-URI, membership-predicate, membership-constant-URI) using ldp:isMemberOfRelation predicate where the constant member resource will be the object of the triple and the newly created resource will be added as the subject of the triple.

In addition, in this example the bugs are associated with the product resource (a non-information resource with a # URI) and not with the product description Direct Container itself. This is done by defining the ldp:membershipResource predicate of the LDP Direct Container as the product non-information resource URI (<?directContainerURI>, ldp:membershipResource, </tracker/product1/#it>). By doing so one can define the subject of the membership triple any resource of interest.
The next example illustrates the behaviour of LDP Direct containers when new resources are created and how aforementioned predicates of the Direct Containers affect the interaction model of LDP.

### 3.2 Creation (POST a resource to an LDP-DC)

Continuing from the previous example, we can report a Bug against the "LDP Demo" product by creating a Bug report (an LDPR representing the bug) under the "LDP Demo" product description LDPC by posting a RDF representation of the Bug report to the LDPC associated with the product description.

The bug report document includes statements about the resource to be created. According the LDP specification, a client can use null relative URI (<> in the request entity body to refer to the resource to be created.
If the creation is successful, the server responds with location of the newly created resource.

EXAMPLE 26: A request for creating a bug

Turtle:
POST /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Content-Type: text/turtle
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .

<> a bt:BugReport;
   dcterms:title "LDP Demo crashes when shutting down."

JSON-LD:
POST /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Content-Type: application/ld+json
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"

{
   "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "bt": "http://example.org/vocab/bugtracker#" },
   "@id": "",
   "@type": "bt:BugReport",
   "dcterms:title": "LDP Demo crashes when shutting down.",
   "dcterms:creator": { "@id": "http://example.org/tracker/users/johndoe"
}

If the creation is successful, the server responds with location of the newly created resource.

EXAMPLE 27: A response of creating new a bug

HTTP/1.1 201 Created
Location: http://example.org/tracker/ldp-demo/bug67
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0

If the creation fails, the server will respond with an appropriate status code depending on the error. If successful, the LDP Demo product description LDPC will have the following representation after the creation of new resource.
EXAMPLE 28: The state of the product LDPC after the bug creation

Turtle:
```turtle
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .

</tracker/ldp-demo/> a ldp:DirectContainer;
    ldp:hasMemberRelation bt:hasBug;
    dcterms:title "Product description of LDP Demo product which is also an LDP-DC";

</tracker/ldp-demo/#it> a bt:Product;
    dcterms:title "LDP Demo";

JSON-LD:
```json
{
    "@context": {
        "dcterms": "http://purl.org/dc/terms/",
        "ldp": "http://www.w3.org/ns/ldp#",
        "bt": "http://example.org/vocab/bugtracker#"
    },
    "@id": "/tracker/ldp-demo/",
    "@type": ["ldp:DirectContainer", "bt:Product"],
    "dcterms:title": "Product description of LDP Demo product which is also an LDP-DC",
    "ldp:contains": [{"@id": "/tracker/ldp-demo/bug3"}, {"@id": "/tracker/ldp-demo/bug4"}, {"@id": "/tracker/ldp-demo/bug67"}],
    "ldp:hasMemberRelation": {"@id": "bt:hasBug"},
    "ldp:membershipResource": {"@id": "/tracker/ldp-demo/#it"}
}
```json

As you can see two new triples are added to the container. That is (<tracker/ldp-demo/>, <ldp:contains>, <tracker/ldp-demo/bug67>) and (<tracker/ldp-demo/#it>, <bt:hasbug>, <tracker/ldp-
demo/bug67>). The former is added in any type of container and the latter is defined by the direct container properties.

The created Bug resource will have the following representation. Note that server has added a server managed property, creation date (dcterms:created), and a default value for the state (bt:isInState) to the Bug in addition to what was POSTed.

EXAMPLE 29: The state of the bug LDPR

@prefix bt: <http://example.org/vocab/bugtracker#> .

</tracker/ldp-demo/bug67> a bt:Bug;
dcterms:title "Product A crashes when shutting down."
; 
dcterms:creator </tracker/users/johndoe>;
dcterms:created "2013-05-05T10:00"^^xsd:dateTime;
bt:isInState "New".

JSON-LD: {
  "@context": {
    "dcterms": "http://purl.org/dc/terms/",
    "bt": "http://example.org/vocab/bugtracker#" },
  "@id": "/tracker/ldp-demo/bug67",
  "dcterms:title": "Product A crashes when shutting down."
, 
  "dcterms:creator": { "@id": "/tracker/users/johndoe" },
  "dcterms:created": {
    "@type": "http://www.w3.org/2001/XMLSchema#dateTime",
    "@value": "2013-05-05T10:00:00"
  },
  "bt:isInState": "New"
}

3.3 Deletion (DELETE on an LDPR associated with an LDP-DC)

This example illustrates the behaviour of a Direct Container when a resource is deleted.
If the delete is successful, the server will respond with a success status code.

```
EXAMPLE 31

HTTP/1.1 204 No Content
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
```

After the deletion, the representation of the container will look like the following
EXAMPLE 32: The state of the product LDPC after the bug deletion

```turtle
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .

<tracker/ldp-demo/> a ldp:DirectContainer;
  ldp:membershipResource </tracker/ldp-demo/#it>;
  ldp:hasMemberRelation bt:hasBug;
  dcterms:title "Product description of LDP Demo product which is also an LDP-DC";

</tracker/ldp-demo/#it> a bt:Product;
  dcterms:title "LDP Demo";
  bt:hasBug <bug4>, <bug67> .

JSON-LD: [{
  "@context": {
    "dcterms": "http://purl.org/dc/terms/",
    "ldp": "http://www.w3.org/ns/ldp#",
    "bt": "http://example.org/vocab/bugtracker#" },
  "@id": "/tracker/ldp-demo/",
  "@type": [ "ldp:DirectContainer", "bt:Product"],
  "dcterms:title": "Product A",
  "ldp:contains": [{ "@id": "/tracker/ldp-demo/bug4"}, { "@id": "/tracker/ldp-demo/bug67" },
  "ldp:hasMemberRelation": { "@id": "bt:hasbug" },
  "ldp:membershipResource":="#it"
} ,
{
  "@context": {
    "bt": "http://example.org/vocab/bugtracker#" },
  "@id":="#it",
  "@type": "bt:Product",
  "bt:hasBug": [{ "@id": "/tracker/ldp-demo/bug4"}, { "@id": "/tracker/ldp-demo/bug67" }
}
]
```

As seen from the LDP Direct Container representation above, both the containment triple
(</tracker/ldp-demo/>, ldp:contains, </tracker/ldp-demo/bug3>) and the membership triple
4. Extended Bug Tracker Example (LDP Indirect containers)

In this next example, we will use the same scenario as in the previous example, but change the container type to a LDP Indirect Container to show the differences between LDP Direct Containers and Indirect Containers and when to use LDP Indirect Containers. Though LDP Direct Containers provide the flexibility to define the constant membership URI (the subject of the membership triple when using ldp:hasMemberRelation or the object of the membership triple when using ldp:isMemberOfRelation) and the membership predicate, when creating members the member derived URI is always the newly created document URL. LDP Indirect containers provide more flexibility by allowing the member derived URI to be any resource; it could be either a non-information resource or a document other than the newly created resource. This done by defining the predicate to look for in the representation of the resource to be created by setting the ldp:insertedContentRelation predicate of the LDP Indirect Container. How this done will be explained in the following examples.

4.1 Navigation and Retrieval (GET on an LDP-IC)

Similar to the previous LDP-DC example, first we will retrieve the representation of the LDP Indirect Container.

EXAMPLE 33: Product lookup request

Turtle:GET /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Accept: text/turtle; charset=UTF-8

JSON-LD:GET /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Accept: application/ld+json; charset=UTF-8

As a response to the GET request, the server responds with the representation of the product description container.
EXAMPLE 34: HTTP response for product lookup
Turtle:HTTP/1.1 200 OK
Content-Type: text/turtle; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#IndirectContainer>; rel="type", <http://example.org/vocab/bugtracker#> .
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Accept-Patch: text/ldpatch
Content-Length: 256
ETag: W/"123456789"

@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

</tracker/ldp-demo/> a ldp:IndirectContainer;
  ldp:membershipResource <#it>;
  ldp:hasMemberRelation bt:hasBug;
  ldp:insertedContentRelation foaf:primaryTopic;
  dcterms:title "Product description of LDP Demo product which is also a LDP-IC";

<#it> a bt:Product;
  dcterms:title "LDP Demo";
  bt:hasBug <bug3#it>, <bug4#it> .

JSON-LD:HTTP/1.1 200 OK
Content-Type: application/ld+json; charset=UTF-8
Link: <http://www.w3.org/ns/ldp#IndirectContainer>; rel="type", <http://example.org/vocab/bugtracker#> .
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH
Accept-Post: text/turtle, application/ld+json
Accept-Patch: text/ldpatch
Content-Length: 278
ETag: W/"123456789"

[
  {
    "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "ldp": "http://www.w3.org/ns/ldp#",
      "bt": "http://example.org/vocab/bugtracker#" },
    "@id": "/tracker/ldp-demo/",
    "@type": [ "ldp:DirectContainer", "bt:Product" ],
    "dcterms:title": "Product description of LDP Demo product which is also a LDP-IC",
    "ldp:contains": [{ "@id": "/tracker/ldp-demo/bug3"}, { "@id": "/tracker/ldp-demo/bug4"}]
  }
]
Now the product container is a LDP Indirect container, which has one main difference: the container has an additional predicate called "ldp:insertedContentRelation". Further, the objects of the containment triples and the membership triples are not the same. While the object of the containment triple is the same (e.g., </tracker/ldp-demo/bug3>, an information resource) the object of the membership triple is now (e.g., </tracker/ldp-demo/bug3#it>, a non-information resource or real world thing). This distinction is because of the ldp:insertedContentRelation definition. How this works will be explained in the next example on creating a new resource.

4.2 Creation (POST a resource to an LDP-IC)

Continuing from the previous example, we can create a new Bug Report against the 'LDP demo' product by creating a Bug Report LDPR under the 'LDP Demo' product description LDPC.

The client POSTs a representation of a Bug Report to the Bug Tracker LDPC.

```json
"ldp:membershipResource": { "@id": "#it" },
"ldp:insertedContentRelation": "foaf:primaryTopic"
},
{
"@context": {
   "dcterms": "http://purl.org/dc/terms/",
   "bt": "http://example.org/vocab/bugtracker#"
 },
"@id": "#it",
"@type": "bt:Product",
"dcterms:title": "LDP Demo",
"bt:hasbug": [{ "@id": "/tracker/ldp-demo/bug3#it"}, { "@id": "/track
```
EXAMPLE 35: A request for creating a bug

Turtle:
POST /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Content-Type: text/turtle

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<> a bt:BugReport;
    foaf:primaryTopic <#it>;
    dcterms:title "Product A crashes when shutting down.";
    dcterms:creator </tracker/ldp-demo/johndoe> .

<#it> a bt:Bug .

JSON-LD:
POST /tracker/ldp-demo/ HTTP/1.1
Host: example.org
Content-Type: application/ld+json

[    
    {      
        "@context": {        
            "dcterms": "http://purl.org/dc/terms/",
            "ldp": "http://www.w3.org/ns/ldp#",
            "bt": "http://example.org/vocab/bugtracker#",
            "foaf": "http://xmlns.com/foaf/0.1/"  
        },      
        "@id": "",      
        "@type": "bt:BugReport",
        "foaf:primaryTopic": { "@id": "#it" },
        "dcterms:title": "Product A crashes when shutting down.",
        "dcterms:creator": { "@id": "/tracker/ldp-demo/johndoe" }    
    },    
    {      
        "@context": {      
            "bt": "http://example.org/vocab/bugtracker#"  
        },      
        "@id": "#it",
        "@type": "bt:Bug"    
    }    
]
One thing to note is that the representation of the resource to be created contains a triple (<> , foaf:primaryTopic , <#it>). If the create request is successful, the server responds with location of the created resource.

```plaintext
EXAMPLE 36: A response of creating new a bug

HTTP/1.1 201 Created
Location: http://example.org/tracker/ldp-demo/bug67
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Content-Length: 0
```

If the creation fails, the server will respond with an appropriate status code depending on the error. After the resource is creation, the Product A LDPC will have the following representation.
EXAMPLE 37: The state of the product LDPC after the bug creation

Turtle:

```turtle
@prefix ldp: <http://www.w3.org/ns/ldp#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix bt: <http://example.org/vocab/bugtracker#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

</tracker/ldp-demo/> a ldp:IndirectContainer;
  ldp:membershipResource </tracker/ldp-demo/#it>;
  ldp:hasMemberRelation bt:hasBug;
  ldp:insertedContentRelation foaf:primaryTopic;
  dcterms:title "Product description of the LDP Demo product which is also a LDP-IC";

<#it> a bt:Product;
  dcterms:title "LDP Demo";
  bt:hasBug <bug3#it>, <bug4#it>, <bug67#it> .

JSON-LD:

```json
[
  {
    "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "ldp": "http://www.w3.org/ns/ldp#",
      "bt": "http://example.org/vocab/bugtracker#",
      "foaf": "http://xmlns.com/foaf/0.1/" },
    "@id": "/tracker/ldp-demo/",
    "@type": ["ldp:IndirectContainer", "bt:Product"],
    "dc:title": "Product description of the LDP Demo product which is also a LDP-IC",
    "ldp:contains": [{"@id": "/tracker/ldp-demo/bug3"}, {"@id": "/tracker/ldp-demo/bug4"}, {"@id": "/tracker/ldp-demo/bug67"}],
    "ldp:hasMemberRelation": {"@id": "bt:hasBug"},
    "ldp:membershipResource": {"@id": "/tracker/ldp-demo/#it"},
    "ldp:insertedContentRelation": {"@id": "foaf:primaryTopic"}
  },
  {
    "@context": {
      "dcterms": "http://purl.org/dc/terms/",
      "bt": "http://example.org/vocab/bugtracker#" },
    "@id": "/tracker/ldp-demo/#it",
    "@type": "bt:Product",
    "dcterms:title": "LDP Demo",
    "bt:hasBug": [{"@id": "/tracker/ldp-demo/bug3"}, {"@id": "/tracker/ldp-demo/bug4"}, {"@id": "/tracker/ldp-demo/bug67"}]
  }
]```
As you can see, two new triples are added to the container. That is (</tracker/ldp-demo/>, <ldp:contains>, </tracker/ldp-demo/bug67>) and (</tracker/ldp-demo/#it>, <bt:hasbug>, </tracker/ldp-demo/bug67#it>).

5. Security

It is not the focus of the Linked Data Platform WG to provide security mechanisms for read/write Linked Data applications; since LDP builds on HTTP, it can re-use any mechanism defined for HTTP. Though most of the security mechanisms that are applicable to general web applications are equally applicable to Linked Data applications, there is still some space to build security mechanisms specific to Linked Data applications by leveraging the Linked Data technologies and providing concrete security requirements for Linked Data applications. In this context, LDP WG has started to create a WG note on Access Control which aims to produce use cases for security scenarios of LDP applications that can be used as the input to a later initiative that will be focused on developing standard security mechanisms for LDP applications.

6. LDP Implementations

A list of implementations that plan to be LDP compliant is available in the LDP Implementations wiki page. The Linked Data Platform 1.0 - Implementation Reports provide the coverage of the specification by each LDP implementation.

7. What To Read Next

The primer only provide an overview of the Linked Data Platform specifications. LDP WG has produced following documents that contribute to the Linked Data Platform specification.

- [Linked Data Platform Use Cases and Requirements](LDP-UCR)
- [Linked Data Platform 1.0 specification](LDP)
- [LDP Best Practices and Guidelines](LDP-BP)
- [Linked Data Platform 1.0 Test Cases](LDP-TESTS)

A. Acknowledgements
Many thanks to John Arwe (IBM), Ashok Malhotra (Oracle), and Henry Story (Apache Software Foundation), Andrei Sambra (MIT) for their thorough review on the LDP Primer document and proposed corrections. We would also like to thank all members of the Linked Data Platform WG for the valuable feedback.

B. Change History

This section is non-normative.

The change history is up to the editors to insert a brief summary of changes, ordered by most recent changes first and with heading from which public draft it has been changed from.

- 2014-06-16 - Addressing the comments and feedback provided by Ashok, John, and Henry.
- 2013-08-05 - Providing JSON-LD representations of the examples.
- 2013-07-03 - Moving the content from the wiki to the note.

C. References

C.1 Informative references

[LDP]
Linked Data Platform 1.0. Steve Speicher; John Arwe; Ashok Malhotra. W3C. 26 February 2015. W3C Recommendation. URL: https://www.w3.org/TR/ldp/

[LDP-BP]

[LDP-TESTS]
Linked Data Platform 1.0 Test Cases. Raúl García-Castro; Fernando Serena; Steve Speicher. W3C. W3C Working Draft. URL: http://www.w3.org/2012/ldp/hg/tests/ldp-testsuite.html

[LDP-UCR]
Linked Data Platform Use Cases and Requirements. Steve Battle; Steve Speicher. W3C. 13 March 2014. W3C Note. URL: https://www.w3.org/TR/ldp-ucr/

[LINKED-DATA]
Linked Data Patterns

A pattern catalogue for modelling, publishing, and consuming Linked Data

Leigh Dodds

Ian Davis
Linked Data Patterns: A pattern catalogue for modelling, publishing, and consuming Linked Data
Leigh Dodds
Ian Davis

Publication date 2012-05-31

Abstract


This work is licenced under the Creative Commons Attribution 2.0 UK: England & Wales License. To view a copy of this licence, visit http://creativecommons.org/licenses/by/2.0/uk/.

Thanks to members of the Linked Data mailing list [http://lists.w3.org/Archives/Public/public-lod/] for their feedback and input, and Sean Hannan [http://twitter.com/MrDys/] for contributing some CSS to style the online book.
# Table of Contents

1. Introduction ................................................................................................................... 1
   Overview .......................................................................................................................... 1

2. Identifier Patterns ............................................................................................................ 4
   Hierarchical URIs ......................................................................................................... 4
   Literal Keys .................................................................................................................... 5
   Natural Keys .................................................................................................................. 6
   Patterned URIs ............................................................................................................. 6
   Proxy URIs ..................................................................................................................... 7
   Rebased URI .................................................................................................................. 8
   Shared Keys ................................................................................................................... 9
   URL Slug ....................................................................................................................... 10

3. Modelling Patterns ....................................................................................................... 12
   Custom Datatype ......................................................................................................... 13
   Index Resources .......................................................................................................... 14
   Label Everything ......................................................................................................... 15
   Link Not Label ............................................................................................................. 16
   Multi-Lingual Literal ................................................................................................. 18
   N-Ary Relation .......................................................................................................... 19
   Ordered List ................................................................................................................. 20
   Ordering Relation ....................................................................................................... 21
   Preferred Label .......................................................................................................... 22
   Qualified Relation ....................................................................................................... 23
   Reified Statement ....................................................................................................... 25
   Repeated Property ...................................................................................................... 26
   Topic Relation ............................................................................................................ 27
   Typed Literal .............................................................................................................. 28

4. Publishing Patterns ....................................................................................................... 30
   Annotation .................................................................................................................... 30
   Autodiscovery .............................................................................................................. 31
   Dataset Autodiscovery ............................................................................................... 32
   Document Type ........................................................................................................... 33
   Edit Trail ....................................................................................................................... 34
   Embedded Metadata ................................................................................................. 35
   Equivalence Links ...................................................................................................... 36
   Link Base ..................................................................................................................... 37
   Materialize Inferences ............................................................................................... 38
   Primary Topic Autodiscovery ................................................................................... 39
   Progressive Enrichment ........................................................................................... 40
   See Also ..................................................................................................................... 41
   Unpublish .................................................................................................................... 41

5. Data Management Patterns ........................................................................................ 44
   Graph Annotation ....................................................................................................... 44
   Graph Per Aspect ....................................................................................................... 47
   Graph Per Resource .................................................................................................. 49
   Graph Per Source ...................................................................................................... 51
   Named Graph ............................................................................................................. 53
   Union Graph ............................................................................................................... 54

6. Application Patterns .................................................................................................... 57
   Assertion Query ........................................................................................................... 57
   Blackboard .................................................................................................................. 58
   Bounded Description ................................................................................................. 59
Chapter 1. Introduction

Abstract

There are many ways to help spread the adoption of a technology, and to share skills and experience amongst a community of practitioners. Different approaches work well for communicating different kinds of knowledge. And we all individually have a preferred means of acquiring new skills, or getting help with a specific problem. Reference manuals, tutorials, recipes, best practice guides and experience reports all have their role. As do training courses, mentoring, pair programming and code reviews.

This book attempts to add to the steadily growing canon of reference documentation relating to Linked Data. Linked Data is a means of publishing "web-native" data using standards like HTTP, URIs and RDF. The book adopts a tried and tested means of communicating knowledge and experience in software development: the design pattern. The book is organized as a pattern catalogue that covers a range of different areas from the design of web scale identifiers through to application development patterns. The intent is to create a ready reference that will be useful for both the beginner and the experienced practitioner alike. It’s also intended to grow and mature in line with the practitioner community.

Overview

Why A Pattern Catalogue?

Design patterns have a number of benefits:

• Patterns have a well-defined structure that encourages focus on the essential knowledge being communicated. This makes them accessible and easy to consume.

• Patterns encourage discussion of related and complementary approaches. Design decisions are rarely clear cut. A focus on specifics is useful for understanding trade-offs

• Patterns provide a name for a particular design decision or solution. Collectively they build a lexicon that encourages clearer communication between practitioners

The authors have successfully applied design patterns in their software development activities. The approach seemed well suited to teasing out some of the experiences and lessons they have learnt through working with Semantic Web technologies; the rigour of a pattern approach also helped the authoring.

...And Why a Book? Why Not a Blog or a Wiki?

This is something that we wrestled with for a long time. Our goal is that this book should ultimately reflect the collective experience of the Linked Data community and we want to encourage participation and feedback. You can use the dataincubator.org [http://dataincubator.org] mailing list to discuss the book and debates its contents. We're also hoping to include your submissions, corrections and edits in the future.

But while we want this book to grow as a participatory activity we (particularly Leigh) felt that an editorial layer would be a useful addition to this process. Helping firm up the naming, communication and organisation of the pattern catalogue as it develops.

We also encourage the community, if they find a design pattern approach to be useful, to publish and share their own patterns using whatever mechanism feels right for them. A thousand flowers, etc. The OntologyDesignPatterns.org [http://OntologyDesignPatterns.org] wiki provides one forum for helping to contribute to this effort.
What's Not Covered?

This book isn't a primer on RDF or OWL. There are already plenty of good sources of introductory material on the technologies discussed here. The book makes the assumptions that you have some basic understanding of RDF, RDF Schema and possibly OWL. The examples are given in Turtle syntax, so you should be familiar with that syntax too.

If you're looking for a deeper introduction to modelling with RDF Schema and OWL then you should read Semantic Web for the Working Ontologist [http://workingontologist.org/]. It's a great book that will give you a thorough understanding of how to apply the technologies. We're hoping that this work is in some sense a companion piece.

How the Catalogue Is Organized

The catalogue has been broken down into a number of separate chapters. Each chapter collects together patterns that have a common theme.

- Chapter 2, Identifier Patterns
- Chapter 3, Modelling Patterns
- Chapter 4, Publishing Patterns
- Chapter 6, Application Patterns

The catalogue also includes a few patterns that arguably aren't patterns at all, they're similar features of the RDF model; Typed Literal for example. We decided to include these for the sake of helping to document best practices. There are plenty of examples and material on some of these basic features but they're often overlooked by both experienced and new practitioners. So we've opted to document these as patterns to help draw attention to them.

Examples

The pattern catalogue includes a number of examples. Snippets of code, data, or queries that help to illustrate a specific pattern. Code examples are shown in this font.

Where we should examples of RDF data, we have used the Turtle [http://www.w3.org/TeamSubmission/turtle/] syntax for RDF because its more concise and readable than RDF/XML. We have preferred to use prefixed names for standard RDF properties and classes but, for clarity, have not always included the declarations of these prefixes in the examples. This allows the example to focus on the particular usage being demonstrated.

Unless otherwise stated, assume that when you're looking at some Turtle we've declared the following prefixes:

```turtle
@prefix ex: <http://www.example.org/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```
In the Data Management Patterns section of the book a number of the examples use the TRiG [http://www4.wiwiss.fu-berlin.de/bizer/trig/] syntax, which is an extension of Turtle that supports Named Graphs.
Chapter 2. Identifier Patterns

Abstract
The single most important part of the Linked Data approach is the adoption of web-scale identifiers (URIs) to identify things of interest: people, events, places, statistical observations, colours. Anything that we want to publish data about on the web needs to have a URI, allowing it to be referenced, browsed and linked using existing web tools. The existing tools of the web of documents are already designed to work well with things that have URIs. We can "like" them, discuss them, and refer to them in documents.

In RDF we capture data as statements about resources. RDF allows resources to have global identifiers or to be un-named "blank nodes". While blank nodes may offer flexibility for some use cases, in a Linked Data context blank nodes limit our ability to collaboratively annotate data. A blank node cannot be the target of a link and we can't annotate it with new information from new sources. As one of the biggest benefits of the Linked Data approach is that "anyone can say anything anywhere", use of blank nodes undermines some of the advantages we can gain from wide adoption of the RDF model. Even within the closed world of a single application dataset, use of blank nodes can quickly become limiting when integrating new data.

Successful publishing of Linked Data requires the careful selection of good, clean, stable URIs for the resources in a dataset. This means that the most important first step in any Linked Data project is deciding on an appropriate identifier scheme: the conventions for how URIs will be assigned to resources. This is not to say that conventions won't evolve or be extended over time, but some upfront thought about identifiers is always beneficial.

This chapter introduces a collection of design patterns that promote some principled ways to assign identifiers within a dataset. All of these patterns are in wide use today and so are tried and tested in the field.

Many of these patterns are also prevalent in modern web frameworks and are more generally applicable to URL design for web applications.

Hierarchical URIs

How should URIs be assigned to a group of resources that form a natural hierarchy?

Context
It is often the case that a collection of resources may form a natural hierarchy. E.g. the chapters within a book, or the departments within an organization. Reflecting this strict hierarchy within the URI structure makes those URIs more hackable allowing users/developers to "navigate" up the hierarchy by pruning the URI.

Solution
Where a natural hierarchy exists between a set of resources use Patterned URIs that conform to the following pattern:

:collection/:item/:sub-collection/:item

E.g. in a system which is publishing data about individual books and their chapters, we might use the following identifier for chapter 1 of a specific book:

/books/12345/chapters/1
Identifier Patterns

The /chapters URI will naturally reflect to the collection of all chapters within a specific book. The /books URI maps to the collection of all books within a system, etc.

Example(s)

The discogs dataset in dataincubator uses hierarchical uris of the form:

http://discogs.dataincubator.org/release/22530/track/1-01

Discussion

This technique is best suited to scenarios where the items in the sub-collections (chapters) are always associated with a single parent item. Other relationships might exist, e.g. the chapter may be included in another but the chapter is always associated with at least one book: they do not exist in isolation. In circumstances where this doesn't hold true, then it is best to just use simple Patterned URIs.

The same applies to circumstances where the hierarchy may change over time.

Related

• Patterned URIs

Literal Keys

How do we publish non-global identifiers in RDF?

Context

The Natural Keys pattern encourages the creation of URIs from existing non-global identifiers. While this provides a way to begin identifying a resource so that we can describe it in RDF, it does not address the issue of how to publish these existing identifiers. Nor does it address situations where natural keys change over time, e.g. the move from ISBN-10 to ISBN-13 in the publishing world.

Solution

Create a custom property, as a sub-class of the dc:identifier property for relating the existing literal key value with the resource.

Example(s)

The nasa dataset in dataincubator uses Patterned URIs based on the NSSDC international designator, but includes these as literal values associated with each spacecraft using a custom property.

Discussion

While hackable URIs are a useful short-cut they don't address all common circumstances. For example different departments within an organization may have different non-global identifiers for a resource; or the process and format for those identifiers may change over time. The ability to algorithmically derive a URI is useful but limiting in a global sense as knowledge of the algorithm has to be published separately to the data.
By publishing the original "raw" identifier as a literal property of the resource we allow systems to look-up the URI for the associated resource using a simple SPARQL query. If multiple identifiers have been created for a resource, or additional identifiers assigned over time, then these can be added as additional repeated properties.

For systems that may need to bridge between the Linked Data and non-Linked Data views of the world, e.g. integrating with legacy applications and databases that do not store the URI, then the ability to find the identifier for the resource provides a useful integration step.

Related

• Patterned URIs
• Natural Keys

Natural Keys

*How can we create unique URIs from data that already has unique identifiers?*

Context

It is often the case that a group of resources already have a unique identifier. This might be a local system identifier derived from, e.g. a database key; or a global non-URI identifier, e.g. an ISBN.

Solution

Mint URIs that are algorithmically derived from the existing non-URI identifier. This can be as simple as concatenating the existing identifier or key with a suitable base URI. The existing identifier may need to be URL encoded before creating the URI. It is common to combine this technique with Patterned URIs.

Example(s)

The BBC programmes website uses URIs that are derived from its existing "programme id" or pid.

Discussion

Where resources are already associated with existing keys, it is likely that the creation and management of those identifiers will already be supported by a specific technology or process. There is a need to be able to create global URI based identifiers for these resources without creating unnecessary additional overheads in creating entirely new identifiers and/or mapping between URIs and existing keys.

By deriving the URI from the natural key for the identifier we avoid the need to create a new process for assigning identifiers and largely eliminate the need to have a mapping between the two identification systems.

Related

• Patterned URIs

Patterned URIs

*How can we create more predictable, human-readable URIs?*
**Context**

Clean, clear URIs can be easier to remember and easier for developers to work with. This is especially true if the URIs can be algorithmically constructed or follow a common pattern. This allows URIs to be constructed or hacked in order to create new entry points into the dataset, e.g. determining the URI for a collection of similar resources based on knowledge of a single example URI.

**Solution**

Create URIs that follow a simple naming pattern. For applications generating Linked Data one technique for building patterned URIs is to use the pluralized class name as part of the URI pattern.

For example if an application will be publishing data about book resources, which are modelled as the `rdf:type ex:Book`. One might construct URIs of the form:

/books/12345

Where /books is the base part of the URI indicating "the collection of books", and the 12345 is an identifier for an individual book.

If multiple classes share a common base class, then it is also possible to use the name of the common base class, rather than generating separate URIs for each derived type

**Example(s)**

The BBC website uses /programmes to group together URIs that relate to series, brands and episodes, all of which are subclasses of the `rdf:type po:Programme`

**Discussion**

There are clear benefits from having human-readable, hackable URIs. This solution achieves that by ensuring the same naming scheme that applies to the underlying data also applies to the URIs. This provides a clear relation between the URI and the type of thing that it describes.

**Related**

- Hierarchical URIs
- Natural Keys

**Proxy URIs**

_How do we deal with lack of standard identifiers for third-party resources?_

**Context**

While it is a goal to reuse identifiers across datasets wherever possible, this is often difficult to achieve in practice. An authority for some specific data, e.g. ISO, may not have assigned unique URIs to resources in their dataset. Datasets also appear online at different times, making reuse difficult until more authoritative data appears and convergence happens on common identifiers. In these circumstances, how should identifiers be created for these third-party resources.
Solution

Treat third-party resources identically to those in your own data and assign them URIs within your domain.

Example(s)

There is still no agreed standard way of generating URIs for Internet Media Types. IANA have adopted RDF for publishing descriptions of registered media types. A data set containing descriptions of images may therefore use locally minted URIs for those media types:

```
ex:anImage a foaf:Image;
   dc:format <http://www.example.org/media-types/image/jpeg>
```

Discussion

A publisher should focus on their immediate goal of opening up their data, ensuring that the published data is internally consistent and has identifiers for all key concepts. If existing identifiers exist then these should be reused. Where they don't then new locally minted URIs should be created from Shared Keys.

Once the data has been published, some alignment can take place within a community to achieve agreement on standard URIs for shared identifiers. One approach for achieving this alignment is to publish Equivalence Links.

Related

- Shared Keys
- Equivalence Links

Rebased URI

_How can we construct one URI based on another?_

Context

Sometimes when generating a Patterned URI the key that we have for as URL Slug is not a simple literal value, but instead another URI. For example this can occur when generating a new Named Graph URI for a resource, or when defining a service URL for a URI Resolver.

Solution

Rewrite the original URI to use a new, predictable base URI using one of the options described below.

Example(s)

An application needs to generate a new Named Graph URI for a resource URI of `http://example.org/document/1`. The application uses a regular expression to rewrite the original URI to a new base, e.g. `http://graphs.example.org/document/1`.
Discussion

URL rewriting is a common feature of all web servers and most web frameworks. Rewriting is normally carried out using regular expressions to match and replace portions of the original URI with some standard replacement text.

URL rewriting is used in several Linked Data services in order to create new URLs. This is typically to support URI resolution for remote (RDF) resources.

Several different approaches seem to be in use. The following examples all show a rewrite for this URI: http://example.org/document/1. Each example notes an example service that uses the approach:

- Simple Prefixing (URIBurner): http://service.example.com/resolve/http://example.org/document/1. The original URI is simply appended to a new base URL. Has the advantage of working with any protocol.

- Prefixing, No Protocol (Triplr): http://service.example.com/resolve/example.org/document/1. The original URI is simply appended to a new base URL after first removing the protocol (e.g. "http://"). Server will need to assume the http protocol if de-referencing the URI or reversing the rewrite.

- Prefixing, With Delimiter (Callimachus): http://service.example.com/resolve;http://example.org/document/1. The original URI is simply appended to a new base URL which ends with a semi-colon.

- Prefixing, As Parameter (SPARQL Graph Protocol): http://service.example.com/resolve?uri=http://example.org/document/1. The original URI is simply appended to a new base URL as a query string parameter.

- Rewritten Authority: http://data.example.org/document/1. The authority component of the URI is rewritten to create a new base URI. This approach might lead to URI clashes unless the input URIs have a predictable structure.

Related

- URI resolver

Shared Keys

How do we simplify the inter-linking of datasets?

Context

It is common to carry out inter-linking of datasets as a separate activity following the initial modelling and data conversion exercises have been completed. How can the final stage of inter-linking be simplified?

It is also common that within a specific domain there will be a set of non-web identifiers that are standardised across different applications and publishers. How can the inter-linking of those datasets be simplified, encouraging convergence on web identifiers?

Solution

Create Patterned URIs by applying the Natural Keys pattern, but prefer public, standard identifiers rather than internal system specific codes.
Example(s)

The BBC have created URIs for artists that are algorithmically related to the MusicBrainz URIs using a common Shared Key. MusicBrainz URIs are Patterned URIs built from a "MusicBrainz ID", e.g. a74b1b7f-71a5-4011-9441-d0b5e4122711. The MusicBrainz and BBC URIs are shown below:

http://www.bbc.co.uk/music/artists/a74b1b7f-71a5-4011-9441-d0b5e4122711
http://musicbrainz.org/artist/a74b1b7f-71a5-4011-9441-d0b5e4122711

Discussion

Predictable URIs structures make it easy for application developers and toolkits to build URIs from simple templates. By using URIs that are constructed from public identifiers, that already have scope outside of the immediate application, it increases the ease with which inter-linking can take place. For example, the pattern may avoid the need to look-up URIs in a SPARQL endpoint, allowing a developer to simplify use a URI template.

The Shared Keys pattern is best suited to situations where the shared identifiers are stable and rarely change.

Related

- Patterned URIs>
- Natural Keys

URL Slug

How can we create URLs from arbitrary text or keywords?

Context

When generating a URI for an identifier we may not always have a simple numeric Natural Key for a resource. In some cases, e.g. for documents, keywords, categories, we may only have a title or name.

Solution

Apply the Patterned URI pattern to create a common root URI and generate a simple URL "slug" by from the available text or keywords.

There are several potential algorithms for normalising a string to create a URL slug. A typical algorithm would be:

- Lowercase the string
- Remove any special characters and punctuation that might require encoding in the URL
- Replace spaces with a dash

The original text is then preserved by using it to label the resource.
Example(s)

For example if we are generating a URI for a category called "Heavy Metal" we might generate a URI as follows:

```html
#Generate a patterned URI with a simple URL slug from "Heavy Metal"
<http:www.example.org/category/heavy-metal>
  rdfs:label "Heavy Metal"
```

Discussion

Clean URLs are easier to work with. If a dataset (or any website) has an inconsistent policy about the casing and encoding used in URLs then it can be more difficult to generate links to that dataset.

The use of URL slugs is prevalent in a number of different web frameworks already, e.g. blogging platforms. So this approach is already well-deployed.

Normalization of text strings can be problematic if it results in the same URL slug being generated for different resources. In some systems names and titles may be unique so this might not be an issue. Using patterned URIs to divide the URI space into different "sections" for different types of resources can help avoid unwanted collisions. Obviously in some cases collisions might be useful, e.g. to ensure that user input, which might vary in casing, is normalized into a standard form.

Related

- Patterned URI
- Natural Key

Further Reading

- Slug (web publishing) on Wikipedia
Chapter 3. Modelling Patterns

Abstract

Creating a model of some domain for publishing data as Linked Data is, fundamentally, no different to any other form of modelling exercise: we need to understand the core entities and their attributes, including their relationships to one another. RDF has a core model that is based on the decades-old Entity-Attribute-Value (EAV) approach that underpins many different information management approaches and technologies. What is different and new for many people is the context in which that modelling is carried out.

When we are creating an Entity-Relationship (ER) or Object-Oriented model we are typically doing so in order to support the needs of a particular application. We have some idea about what data the application needs to store and we tailor the model accordingly. We also typically optimise the model for the needs of that application. In particular, when translating a logical ER into a physical database schema, we often need to introduce new entities, e.g. to capture many-many relationships in order to be able to express the model within the restrictions imposed by a relational database.

In a Linked Data context we approach a modelling exercise in a different way.

Firstly we may be able to avoid the exercise completely. Using RDF Schema and OWL it is possible to share vocabularies (or ontologies, or schemas, whatever your preferred term) on the web, just as we share data. This means that communities of interest can collaborate around a data model and then extend it where necessary. With semantic web technologies we can better crowd-source and share our data models.

It is often the case that a combination of existing vocabularies will adequately cover a domain. Where there is a short-fall we can define small extensions, e.g. new properties or types, that extend the model for our purposes.

The second key difference is that we focus on modelling the domain itself and set aside the immediate needs of the application. By focusing on the model rather than the application, we are more likely to be able to extend and enrich the model at a later date. Thinking about the entities and their relationships, rather than our application code, results in a stronger model. And one that may be more likely to be of use to others.

Of course a model must ultimately support the kinds of data we want to capture or manipulate in our application. The ability to populate and query a model is a good test of its fit for purpose, but application requirements alone shouldn’t guide our modelling.

This is not the same as saying that all modelling must be done up front. Modelling for Linked Data can be agile and iterative. We should just take care to think about the likely areas of extension, as the Link Not Label pattern illustrates.

Thirdly, and finally, the other important way in which RDF modelling differs from other approaches is that there is no separate physical data model: the logical model is exactly how our data is stored. An RDF triple store is "schema-free" in the sense that we don't have to define the physical layout (e.g. tables, columns) of our database. Any RDF triple store can store any RDF data expressed using any RDF vocabulary.

This makes RDF very amenable for rapid, iterative application development. Importantly it also means that everyone involved in a project team is working off the same domain model; the same mental model of how the resources in a dataset are described and how they relate to one another.

This chapter introduces a number of RDF modelling patterns that are useful in a variety of different use cases. Several of the patterns are really illustrations of how to use particular features of RDF, e.g. annotation of literals with data types or languages. We’ve included these within the overall patterns collection as we feel that presenting them as solutions to specific questions, e.g. "How can internationalized text be expressed in RDF?" may help address what are frequently asked questions by new users.
Custom Datatype

A data model contains structured values that don't correspond to one of the pre-existing XML Schema datatypes

Context

Some applications may have pre-defined custom datatypes for describing the structure of literal values. It is useful to preserve this type information when publishing it as RDF in order to allow applications to handle them correctly.

Solution

Create a URI to identify the custom datatype and use that URI when creating Typed Literals

Example(s)

#Define a URI for different forms of shipping codes
<http:www.example.org/datatype/FedexShippingCode>
  rdfs:label "Fedex Shipping Code".
<http:www.example.org/datatype/UPSShippingCode>
  rdfs:label "UPS Shipping Code".

#Indicate the form of shipping code with a Typed Literal
_:delivery1

Discussion

RDF does not place any limits on what datatypes might be associated with a literal. The model recommends using a standard type library, like XML Schema, wherever possible but the only real constraint is that datatypes should be identified by a URI. Creating a new URI for an application or domain specific datatype allows the type system to be easily extended. The datatype URI can be annotated with additional RDF statements, e.g. a label or description, to describe its usage. With sufficient detail on the data type, reasoners may be better equipped to draw new knowledge from a data model or highlight inconsistencies.

The key limitation to custom datatypes is that SPARQL query engines will not understand how to compare values of that type. Some SPARQL query processors may provide support for understanding the range of types it understands.

Note that a typed literal with a custom datatype can also be modelled as a sub-property. Continuing the above example, instead of defining a new type for each shipping code, the model could have instead been structured as:
<http:www.example.org/def/post/fedexShippingCode>
  a rdfs:Property;
  rdfs:subPropertyOf <http:www.example.org/def/post/shippingCode>
  rdfs:label "Fedex Shipping Code".

#Use the derived property:
_:delivery1
  ex:fedexShippingCode "1234-568".

The advantages of this alternate approaches are:

• Simpler querying of data by allowing the use of triple patterns, rather than FILTERs for extracting data of a common type.

• More explicit, fine-grained semantics

But there are several disadvantages to this approach:

• Increases the number of very similar properties, requiring redundancy of data or reasoning to allow applications to treat them generically

• Doesn't address the core requirement to indicate the lexical form of a structured value

With this in mind the general recommendation is to:

• Use a custom datatype to label particular types of structured value that share a common lexical form. These values may be associated with a broad range of different properties. Processing applications may want to implement a common set type conversions or display options for the values.

• Use a sub-property in all other cases

Note that these options aren't always mutually exclusive. It might be useful in some scenarios to have an explicit property for associating a general class of code, identifier, or other Literal Key with a resource, but also assign a specific datatype to the identifier as a cue for applications and reasoners

Related

• Typed Literal

• Literal Key

Index Resources

How can ordered collections or indexes be published as RDF?

Context

It is often the case that a web application will provide its users with a number of alternative ways to navigate. This might be an A-Z list of resources, or a list ordered based on creation date or "interestingness". These collections are useful resources in their own right that should be published as Linked Data.
Modelling Patterns

Solution

Create resources for each "index" and associate the index with the items that it contains. An rdf:List or rdf:Seq is best used here as these structured include the notion of ordering that is an essential aspect of an index.

Example(s)

The BBC /music website publishes an A-Z list of artists. But this does not use a sequence to indicate ordering.

Discussion

By creating separate resources and links between these indexes and their contents we avoid the need to publish the algorithms that are used to generate the indexes, we can instead allow machines to navigate the collection just as a human user would do. This approach is particularly useful to ensure that data can be crawled by a semantic web search engine.

If a collection may be unmanageably large, then it can be decomposed into smaller groupings. E.g. a large A-Z index may be decomposed into smaller collections: A-C, D-F, etc.

By using an rdf:List or rdf:Seq to order the association between the index and its contents we retain the notion of ordering. Where the index is a permanent collection, with all contents known at the time of publishing, then an rdf:List is the best structure. Otherwise use an rdf:Seq. These RDF data structures can easily be generated and maintained using an asynchronous indexing system, avoiding the need to rebuild all the indexes whenever the underlying data changes. However for a large number of different indexes or variations the time spent building these indexes can be expensive.

Related

- Hierarchical URIs
- Composite Descriptions

Label Everything

*How can we ensure that every resource has a basic human-readable name?*

Context

A dataset may have a number of different entities, some of which are simple, e.g. people or organizations, whereas others are more conceptual or complex, e.g. an observation made at a particular point in time, under specific conditions. It may not always be clear to a developer, or a user exploring a graph in a browser, what a particular resource represents

Solution

Ensure that every resource in a dataset has an rdfs:label property

Example(s)
ex:Book
  rdfs:label "War and Peace".

ex:WeatherObservation
  rdfs:label "Rainfall measurement from Weather Station 1 recorded by Bob on 17th August 2011";
  ex:rainfall 50;
  ex:date "2011-08-17"^^xsd:date
  ex:location ex:WeatherStation1;
  ex:experimenter ex:Bob.

Discussion

The rdfs:label property is a useful generic property for labelling any type of resource. By using this generic property to label any resource we can ensure that applications can easily discover a useful default label for a specific resource using a well-known property. This is particularly useful for supporting browsing of a dataset, as a browser can look for a default label. Developers can also use the label to assist in debugging queries or exploring a dataset.

Client applications may not always wish to use a provided label instead preferring to construct them based on other criteria. The Preferred Label pattern recommends using the skos:prefLabel property to communicate to clients a preferred label specified by the data publisher.

In some cases both a rdfs:label and a skos:prefLabel (or other specific labelling property such as dcterms:title) might be provided for the same resource. The content of the labels may differ reflecting the slightly different semantics of each property, e.g the rdfs:label might be longer or more descriptive than a shorter skos:prefLabel. If both label properties are provided with the same content, then this is an example of the Materialize Inference pattern: skos:prefLabel is a specialization of rdfs:label.

The importance of applying labels to Linked Data, as well as evidence for the poor adoption of this practice, is given in a paper called "Labels in the Web of Data" by Basil Ell, Denny Vrandečić, and Elena Simperl.

Related

- Preferred Label

Further Reading

- Labels in the Web of Data (Paper)
- Labels in the Web of Data (Video Lecture + Slides)

Link Not Label

How do we model a dataset to maximise benefits of a graph based model?

Context

Most datasets are centred around a few core resources types that are central to that dataset. For example a social network dataset would be centred on people, groups and organizations, whereas a publishing dataset would be centred on authors, publications and publishers.
However every dataset typically has some additional resource types that are less "central". E.g. areas or topics of interest, spoken or print languages, publication formats, etc. Often these resources are overlooked during modelling and are often only represented as simple literal values. This can make it less efficient to query a dataset, e.g. to group resources based on shared characteristics (e.g. everyone with same interests, all hardback books). It can also limit the annotate these aspects of a dataset, e.g. in order to add equivalence links.

Many common modelling approaches or industry standard models often re-inforce a less expressive modelling approach. For example many publishing and library standards, while encouraging use of controlled terms and authority files, focus largely on publications as the only important entity in a data model, leaving subject categories and authors as little more than labels associated with a work.

**Solution**

Ensure that all resources in a dataset are modelled as resources rather than simple literal values. Relate resources together to create a richer graph model. Use the literal values as the labels of the new resources.

A good approach is to look for any controlled vocabularies, keywords, tags, or annotations and dimensions in a dataset and model them as resources. Even structured literal values like dates might be more usefully modelled as resources.

**Example(s)**

Example of potential resources that might get overlooked:

- Languages
- Country codes
- Tags, categories, or subject headings
- Gender
- Genres
- Formats

A simple example:

```xml
# Rather than this:
<http://www.example.org/book/1>
   dc:format "Hardback";
   dc:lang "en"
   dc:subject "RDF", "SPARQL".

# Use a richer model:
<http://www.example.org/book/1>
   dcterms:format <http://example.org/formats/hardback>;
   dcterms:lang <http://www.lingvoj.org/lingvo/en>;
   dcterms:subject <http://example.org/category/rdf>;
   dcterms:subject <http://example.org/category/sparql>.
```

<http://example.org/formats/hardback>
rdfs:label "Hardback".

<http://example.org/category/rdf>
  rdfs:label "RDF".

<http://example.org/category/sparql>
  rdfs:label "SPARQL".

#Categories could later be related to other sources
<http://example.org/category/rdf>
  owl:sameAs <http://id.loc.gov/authorities/sh2003010124#concept>;

Discussion

Creating a rich graph of relationships within a dataset will yield more flexibility and value from adopting Linked Data.

For example, as RDF triple stores are optimized for storing and querying relationships and graph patterns, creating resources for common dimensions in a dataset will allow for faster and more flexible querying. By representing these dimensions as resources in their own right, then they can be more easily annotated, e.g. to qualify them with additional data, or relate them to external sources.

In many cases existing resources in publically available datasets can be used instead of creating new URIs. By using resources, and reusing identifiers, it becomes easier to correlate and traverse different datasets. For example it becomes possible to draw in external data to enrich an existing application, e.g. an author profile or related works in another collection.

Related

- Annotation
- Equivalence Links

Further Reading

- Why Resources in Linked Data are Good

Multi-Lingual Literal

*How can internationalized text be expressed in RDF?*

Context

It is increasingly common for data to contain internationalized text, e.g. translated titles for a document. This alternate language text needs to be associated with the relevant resource in a clearly identifiable way.

Solution

Multi-lingual versions of a literal property can be expressed as a simple Repeated Property with the addition of a language code to distinguish between the alternate versions.
Example(s)

_:greeting a ex:LoginGreeting; skos:prefLabel "Hello!"; skos:prefLabel "Hola!"@es.

Discussion

RDF allows a literal value to be associated with a language code. This code indicates the language in which the value of the literal has been expressed. RDF uses the ISO standard language codes, including regional variants. This capability lowers the bar to internationalizing data that is published as RDF. All serializations support this functionality and the SPARQL query language provides several functions for matching and manipulating language codes.

Related

• Repeated Property

N-Ary Relation

How can a complex relation, involving several resources, be modelled as RDF?

Context

An RDF triple expresses a relationship between at most two resources. However some relationships are not binary and involve the equal participation of several different resources. This is most common in the case of events, where the event is a relation between several resources, e.g. a Purchase is an n-ary relationship between a Person, a Product, and a Seller. All of those participants are key to the relationship.

Solution

Create a class for the relationship, and create instances of that resource to relate together the other resources that are involved in the relation.

Example(s)

ex:bob a foaf:Person.
ex:mary a foaf:Person.
ex:conferenceCentre a ex:Building.
ex:legoinc a foaf:Organization.

_:event1 a ex:Conference;
dc:title "Lego Hack Day";
ex:organizer ex:mary;
ex:attendee ex:bob;
ex:sponsor ex:legoinc;
ex:location ex:conferenceCentre.
Discussion

The N-ary Relation pattern is similar to the Qualified Relation pattern as both involve the same basic solution: modelling a relationship as a resource rather than a property. They differ in their context. In the Qualified Relation pattern the desire is to annotate a relationship between two resources, whereas in the N-ary Relation pattern the goal is to represent a complex relation between several resources.

Related

- Qualified Relation

Further Reading

- Defining N-ary Relations on the Semantic Web [http://www.w3.org/TR/swbp-n-aryRelations/]

Ordered List

How do we specify an ordering over a collection of resources?

Context

Sometimes ordering is an important aspect of some collection of data. E.g. the list of authors associated with an academic paper, or the placings on a scoreboard. RDF offers a several different ways to capture ordering across some portion of a graph.

Solution

Use an rdf:List to describe a ordered list of resources.

Example(s)

```xml
<http://www.example.com/docs/1> ex:authors ( 
  <http://www.example.com/author/joe> 
  <http://www.example.com/author/bob> 
).

<http://www.example.com/author/joe> 
  foaf:name "Joe".

<http://www.example.com/author/bob> 
  foaf:name "Bob".
```

Discussion

RDF offers several modelling options for defining collections of resources. Formally these are the RDF Containers (Sequence, Bag, and Alternates) and the RDF Collections (List). For this purposes of this pattern an RDF Sequence and an RDF List are very similar: both describe an ordered list of resources.
Semantically the two structures differ in that a sequence is open ended (i.e. other members may exist, but aren't itemized) while a list is closed (i.e. the members are complete).

In practice though there is no real difference between the structures as data cannot be easily merged into an existing sequence, e.g. to append values. Both also suffer from being poorly handled in SPARQL 1.0: there is no way to query or construct an arbitrary sized list or sequence without extensions; SPARQL 1.1 property paths will remedy the querying aspects.

**Related**

- Repeated Property

**Ordering Relation**

*How can we specify an ordering relationship between two or more resources?*

**Context**

Ordering is important in many data models. Often that ordering can be implicit in the properties of resources, e.g. publication dates of Articles. In some cases the ordering needs to be more explicit, describing a fixed ordering relationship between the resources, e.g. a sequence of events, or stops on a delivery route.

**Solution**

Create ordering properties, e.g. "previous" and "next", for specifying the relationship between the ordered resources.

**Example(s)**

The following example describes the bus route, consisting of a number of bus stops. A stop may be preceded by a previous stop and may be followed by another stop. Custom relations have been defined to identify the previous and next relationships between the bus stops.

```xml
ex:bus10 a ex:Bus;
   ex:route ex:stop1.

ex:stop1 a ex:BusStop;
   ex:number 1;
   ex:nextStop ex:stop2.

ex:stop2 a ex:BusStop;
   ex:number 2;
   ex:previousStop ex:stop1;
   ex:nextStop ex:stop2.

ex:stop3 a ex:BusStop;
   ex:number 3;
   ex:previousStop ex:stop2.
```
Discussion

RDF provides several ways to implicitly and explicitly defining ordering in a dataset. Resources often have literal properties that implicitly define an ordering relationship between resources. E.g. time stamps on events, publication dates, or in the above example the numbers of each of the bus stops. Application code can use a query to extract this implicit ordering from the date. However its otherwise not defined in the model.

Another, more explicit way to define order is by using an RDF collection to create an Ordered List. Using an Ordered List, the above example could have been expressed as:

```protobuf
ex:bus10 a ex:Bus;
  ex:route ( ex:stop1 ex:stop2 ex:stop3 ).
```

Lists are useful when there is no explicit relationship between the resources in the ordered collection. The above example has lost the fact that Bus Stop 2 is preceded by Bus Stop 1, other than their arrangement in a list. An application could still discover this relation using a query, but this can be difficult, and is certainly less explicit than the original example.

An ordering relation provides an explicit relationship between the ordered resources; in effect the ordering relations define a linked list that connects the resources together. Ordering relations are also simple to query and manipulate. For example, it would be very simple to adjust the ordered relation approach to modelling a bus route to add an extra stop, whereas the list based approach is more difficult as the entire list needs to be re-built requiring more changes to the graph.

The trade-offs between using an Ordering Relation versus an Ordered List depends on the specific data model. Continuing the above example, if the ordering of stops is generally fixed, then it is appropriate to using an Ordering Relation. However, if new bus routes are constructed by selecting from the available bus stops then an Ordered List may be more appropriate as there is no longer a fixed relationship between the stops.

Related

- Ordered List

Further Reading

- Defining N-ary Relations on the Semantic Web [http://www.w3.org/TR/swbp-n-aryRelations/#pattern2]

Preferred Label

*How can a simple unambiguous label be provided for a resource?*

Context

There are many different RDF properties that can be used for expressing a label, including generic properties such as rdfs:label and more domain specific labels such as foaf:name. This presents a variety of choices to the data provider, and can be confusing for application authors.
Solution

Use `skos:prefLabel` to publish the preferred display label for a resource

Example(s)

A library system publishes Linked Data about authors. The preferred mechanism for specifying author names is using a normalized form, e.g. Surname, First Name. This preferred label can be specified using `skos:prefLabel`, but the authors full name, and part names can be published using properties from FOAF. A simple Linked Data browser may use the preferred label, whereas an Address Book application browsing the same data may choose to assemble display labels according to user defined preferences, e.g. First Name, Surname.

Discussion

The `skos:prefLabel` property, whilst defined in the SKOS ontology, is a general purpose property that identifies the preferred label for a resource. By standardising on this property for labelling data providers and consumers can converge on a common mechanism for expressing labels of resources.

Having a single preferred property for display labels encourages convergence but doesn't preclude the use of more specific properties for other purposes. For example the literal value of the `skos:prefLabel` property can be formatted for display, leaving other properties, e.g. `rdfs:label`, `foaf:name`, etc to express alternatives or variations in labels or naming. A client that is aware of the meaning of specific predicates may choose to build labels using alternate logic, but having a label unambiguously specified simplifies application development.

Qualified Relation

_How can we describe or qualify a relationship between two resources?_

Context

In some cases relationships need to be qualified or annotated in some way. There are a number of different use cases that required this capability. E.g. to indicate the date when a specific relationship was made, or to indicate its source, or perhaps associate it with a probability or other similar qualifier. RDF only allows binary relations between resources, so how can these additional qualified relations be expressed?

Solution

Create a class for the relationship and create instances of that resource to relate together the resources that are involved in the relation. The relationship resource can then be annotated to qualify the relation further.

Example(s)

Marriage is a relationship that could be modelled as a simple relationship between two people. But that simplistic approach doesn't let us capture the date that the marriage started (or ended). Modelling the marriage as a relationship allows the relationship to be annotated:

```
eg:bob a foaf:Person.
eg:mary a foaf:Person.
```
A diagnosis can be viewed as a relationship between a person and a disease. A diagnosis needs to be qualified with a probability. By creating a class to model the diagnosis explicitly, as well as additional properties for relating the diagnosis to a patient and a disease, it becomes possible to annotate the relationship with qualifying properties:

```turtle
_:bobDiagnosis a ex:Diagnosis;
  ex:patient eg:bob;
  ex:disease eg:measles;
  ex:probability "high";
  ex:diagnostician ex:drhouse.
```

**Discussion**

Modelling relationships as resources works around the limitations of simple binary predicates. Creating a resource for the relationship allows much more flexibility in qualifying or describing the relationships between resources. Any number of additional properties may be used to annotate the relation. When the relationship is between two resources we refer to it as a qualified relation, when it is between several resources, each of which are equally involved in the relationship then we have an N-Ary Relation.

If modelling relationships as classes is useful, then why not use this pattern for all non-trivial relationships in a model? The main reason is that it causes explosion in the number of terms in a vocabulary, e.g. each predicate is replaced with two predicates and a class. A vocabulary can quickly become unwieldy, so the value of the extra modelling structure needs to be justified with clear requirements for needing the extra complexity. As described here, the primary reason is to qualify the relation.

The only alternative to this approach would be to have another independent property whose value is intended to be interpreted alongside one or more other properties of the same resource, e.g.:

```turtle
eg:bob a foaf:Person.
eg:mary a foaf:Person.

eg:bob ex:partner eg:mary.
```
In the above alternative the marriage relationship between two people is expressed using the `ex:partner` relation and the date of the wedding with the separate `ex:weddingDay` property. On the surface this seems simpler but loses flexibility and clarity. Firstly there could be a large number of additional properties associated with the marriage relation, all of these would need to be added to the model, and applications would need to know that this collection of properties were all related in some way (i.e. were about a marriage). It also fails to model divorces and re-marrriages. Adding a relation resource deals with this better as we have greater clarity in the model about the relationship and its specific properties.

There are a number of cases where adding resources into a data model in this way can aid expressivity, understanding when and when not to apply the pattern is an important part of RDF modelling.

**Related**

- N-Ary Relation

**Reified Statement**

*How can we make statements about statements?*

**Context**

The foundation of the RDF model is the triple. This simple structure can be combined to create complex descriptions that can support any kind of data model. But in some circumstances there may be a need to annotate an individual triple, e.g. to indicate when the statement was made, or who by. How can this be achieved?

**Solution**

Use a technique known as reification and model the triple as a resource, with properties referring to its subject, predicate and object.

**Example(s)**

```rdfs
_:ex rdf:type rdf:Statement;
   rdf:subject <http://www.example.com/book/1>;
   rdf:predicate <http://xmlns.com/foaf/0/1/maker>;
   rdf:object <http://www.example.com/author/joe>;
   dc:created "2010-02-13".
```

**Discussion**

Reification is a much maligned technique but has its role to play in RDF modelling. Understanding its limitations allows it to be used where appropriate and avoided when other techniques like Named Graphs are a better fit.

RDF triples cannot be directly annotated. Reification is a modelling approach to dealing with this that involves changing the structure of the model. Each triple that needs to be annotated is instead modelled as
a resource in its own right; an instance of the rdf:Statement class. The statement resource then has subject, predicate, and object properties that describe the original triple; additional properties can then be added to annotate the statement.

The obvious limitation of this approach is that the data model no longer contains a simple set of triples, instead it contains *descriptions of triples*. The triples themselves have not been explicitly asserted. This makes a reified model harder to query, more difficult to manipulate, and at least three or four times larger in terms of number of triples.

In contrast to reification the Named Graph pattern offers a data management approach to annotating triples or sets of triples, by associating them with a URI which can itself be the subject of additional assertions.

So where is reification best used? Current practice seems to suggest that reified statements are best used as a technique for describing changes to the structure of a graph, e.g. statements that have been added or removed, along with additional properties to indicate the time of the change, etc. In other circumstances it is best used with a reasoner or rule engine that can support surfacing the original triples, thereby simplifying querying. But this still comes with a cost of increased storage as well as potentially performance issues.

### Related
- Named Graphs

### Repeated Property

*How can properties that have multiple values be expressed?*

### Context

A resource may have multiple values for a specific property. E.g. a set of keywords associated with a research paper. RDF offers several options for modelling these multi-valued relations. Sometimes these multi-valued relations have an explicit ordering that should be preserved in the data, e.g. the order of authors. In other cases an explicit ordering is not required

### Solution

Simply repeat the property of the resource multiple times.

### Example(s)

For example a research paper may have multiple keywords. The ordering of these keywords is not important, so using the Dublin Core subject property, we can express this multi-valued relation as:

```r
_:doc
    dc:subject "RDF";
    dc:subject "Semantic Web".
```

### Discussion

Repeating properties is the simplest approach to handling multi-valued relations. The alternatives all have their downsides. One alternative would be to use a structured value for the literal, e.g:

```r
_:doc
    dc:subject "RDF";
    dc:subject "Semantic Web".
```
_:doc dc:subject "RDF, Semantic Web".

But structured values limit the ability for applications to query for specific data items, e.g. documents that have a specific keyword. With a structured value a regular expression would be required to test for the values in the literal.

Another alternative would have been to use an RDF Container (Sequence, Bag, or Alt) or Collection (RDF List). Some of these structures imply ordering (Sequence, List) while others (Bag, Alt) don't imply an ordering over their members. This means that the keywords could have been expressed as:

_:doc dc:subject ( "RDF", "Semantic Web" )

There are several downsides to using Containers and Collections. Firstly they can be difficult to use as they can be hard to query. Secondly, when combining data from different sources the Containers and Collections won't be merged together to create a single structure; the lists will remain separate.

On the whole, the simplest way to express multi-valued relations is to use a simple Repeated Property. Collections and Containers are best used only when ordering is significant.

Related

- Ordered List

**Topic Relation**

*How can a web page or document be associated with a resource?*

**Context**

There is often a need to associate a resource with a page or document that provides further, human-readable content about the resource. This information may often be present in a content management system or dataset as "further reading" links. How can these relations be published as RDF, clearly identifying the relationship between the document and the resource being described?

**Solution**

Use the foaf:topic and/or foaf:primaryTopic relationships to associate a resource with a page(s) in which it is featured or discussed.

**Example(s)**

In a dataset describing Paris, some additional further reading links may be available:

<http://www.example.org/place/Paris> a ex:Place;
   skos:prefLabel "Paris"@fr.
Discussion

There will always be a mix of unstructured documents and structured data on the Semantic Web. There are many kinds of relations that could be expressed between a document and a resource, but a very common relation is one of categorisation, e.g. identifying when a document or page has a specific focus or theme. The FOAF topic terms, foaf:topic and foaf:primaryTopic provide a means to link a document directly with a resource which features as a topic in that document.

Other vocabularies provide similar related properties that may be more useful in some scenarios. For example Dublin Core defines the dc:subject property, but this is best used to relate a document to an entry in a controlled vocabulary, e.g. a subject heading, rather than a "real world" resource.

Because the Topic Relation pattern involves making statements about other resources on the web, it is an application of the Annotation pattern.

Related

• Annotation

Typed Literal

*How can a datatype be associated with an RDF literal?*

Context

Data items are generally of a specific type, e.g. a date, floating point value, or decimal. This type information is important for consuming applications as it provides flexibility when querying, formatting, displaying and converting data.

Solution

Always associate a data type with an RDF literal that contains a structured value.

Example(s)

TODO

Discussion

The RDF allows a literal value to be associated with a data type. RDF itself does not have a built-in type system, it defers to the XML Schema datatypes to define a useful common set of data types and their legal lexical values. By using this facility data publishers can ensure that consumers can more easily manipulate and process the published data. Use of standard data types encourages interoperability between systems. It
also supports internationalization of data as client applications can more easily process the value to present it for display in a specific locale

In some cases XML Schema does not define an existing data type. It is therefore common practice to define a Custom Datatype

**Related**

- Custom Datatype
Chapter 4. Publishing Patterns

Abstract

There is an increasingly wide variety of organisations who are publishing Linked Open Data. The growth of the "Linked Data Cloud" is a staple of all Linked Data presentations and charts the early success of the community in boot-strapping an interconnected set of datasets across the web.

There is also a wide variety of ways in which Linked Data is being published, including:

• simple statically generated RDF and HTML files
• RDFa embedded in web application pages
• As an extension to the principled design of a web application that supports a rich variety of views and data formats
• As independent datasets dynamically generated over data stored in triple stores, relational databases or other data sources

In addition data might be published as both a primary or secondary sources, e.g. as an RDF conversion of a data dump available from another organisation. The ongoing challenge for the growth and adoption of Linked Data will be in simplifying getting more data online, e.g. continuous improvement to tools, as well as the introduction of more primary sources that are commitment to publishing high quality, regularly updated data.

Regardless of the source of the data or the means of its publication, there are a number of recurring patterns and frequently asked questions that relate to best practices around data publishing. This chapter documents a number of patterns relating to Linked Data publishing, and in particular how data can be made more discoverable, and over time enriched and inter-linked.

While many of these patterns may have been discovered through the publication of Linked Open Data, they are generally applicable to Linked Data publishing in other contexts, e.g. inside an enterprise.

Annotation

How can data about third-party resources be published as part of a dataset?

Context

Datasets are rarely completely self-contained. They will often contain facts or data about third-party resources. These may be entities from other systems or existing web documents that are part of an external system or website. It should be possible to surface this information alongside the data that originates in the dataset.

Solution

Just publish RDF documents containing the statements about the external resources

Example(s)

Linked Data available from http://www.example.org/author/john when retrieved might contain the following data which contains annotations about two additional resources:
<http://www.example.org/author/john> a foaf:Person.

<http://wiki.example.net/page/UbuntuTips>
  dc:title "Ubuntu Tips";

<http://publisher.example.org/authors/1234>
  owl:sameAs <http://www.example.org/author/john>.

Discussion

With RDF Anyone can say Anything Anywhere, there are no restrictions about who may make statements about a resource; although clearly a processing application might want to pick its sources carefully.

It is entirely consistent with the Linked Data principles to make statements about third-party resources. While in some cases it is useful to use the Proxy URIs pattern, in many cases simply making statements about external resources, whether these are entities (e.g. a person) or documents (e.g. a web page) is sufficient.

Related

• Proxy URIs
• Topic Relation

Autodiscovery

How can people find the underlying linked data for a given web page?

Context

Currently the Web is predominately built from interlinked HTML pages. Linking directly to linked data documents from HTML presents the risk of confusion for non-technical audiences. However the publisher requires that the underlying data be discoverable by linked data aware tools and indexable by search engines.

Solution

When publishing a web page derived from linked data include a link element in the head of the web page pointing to the original data.

<link rel="meta" type="application/rdf+xml" title="Raw Data" href="http://example.com/data.rdf"/>

Example(s)

The FOAF Vocabulary recommends linking a homepage to an equivalent FOAF profile using the link element.
The Semantic Radar Firefox plugin uses the autodiscovery pattern to detect the presence of linked data related to the web page the user is viewing.

Discussion

Until web browsers become fully linked data aware it may not be satisfactory to link directly to linked data pages from the body of an HTML page. HTML provides the link element to allow publishers to include links to information and resources that may be relevant in addition to the main content of the page. Web browsers may choose to display these links in their user interface. Web search engines can use these links to discover additional information that may make their search more relevant for the user.

Related

• Primary Topic Autodiscovery

Dataset Autodiscovery

*How can an application discover the datasets (and any associated APIs) published by a website?*

Context

A Linked Data application might apply the Follow Your Nose pattern to discover additional data about resources in an RDF graph. But the application may need to discover additional metadata about the dataset to which the resource belongs, such as its license. The application may also need access to additional facilities, such as a SPARQL endpoint, in order to extract any additional data it requires.

Solution

Use the Vocabulary of Interlinked Datasets (VoiD) vocabulary to publish a description of a dataset. Make this description available from the standard location at the domain of the website, i.e. /\.well-known/void.

Example(s)

An application discovers the URI http://data.example.org/thing/1 in a dataset. De-referencing that URI will yield a description of that resource. In order to discover a SPARQL endpoint for that dataset, the application performs a GET request on http://data.example.org/\.well-known/void. This URL returns a description of the dataset published at that domain.

Discussion

RFC 5785 defines a means for declaring well known URIs for a website. A registry of well known URIs is maintained by the IETF, and a registration has been made for /\.well-known/void. As the VoiD specification indicates, this URI should return a description of all datasets available from the specified domain.

Providing such a description allows a consuming application to bootstrap its knowledge of available datasets, their licensing, and the means of interacting with them. This can greatly simplify working with unknown data sources. For example an application could discover a SPARQL endpoint or data dump that would allow it to more efficiently harvest some required data.
### Document Type

*How can some context be provided about a set of RDF triples published to the web?*

### Context

While the web of Linked Data is, in its broadest sense, a set of RDF triples, there are often circumstances in which it is useful to describe a smaller grouping of triples. RDF statements are published to the web as documents conforming to a particular syntax, e.g., RDF/XML, Turtle, or XHTML+RDFa. These documents may be directly inter-linked using See Also relations. To enable user agents to select between links it is useful to indicate the type of document which a link is referencing.

### Solution

Define a document type describing a conceptual or physical grouping of triples. Indicate where a specific document is of a particular type, including a Topic Relation such as `foaf:primaryTopic` to relate the document to the resource(s) it is describing.

### Example(s)

```xml
#document type
foaf:PersonalProfileDocument a foaf:Document.

#specific instance of document, with indication of its topic
<http://www.example.org/doc/john> a foaf:PersonalProfileDocument;
  foaf:primaryTopic <http://www.example.org/doc/john#me>.

<http://www.example.org/doc/john#me> a foaf:Person;
  foaf:name "John".
```

### Discussion

XML is a document format and XML schemas describe the valid structure of documents. In contrast RDF is defined in terms of sets of triples and schemas are used to support inferencing and description of data structures. It is often useful to describe specific collections of triples. For example within a triple store it is often useful to group triples into Named Graphs. These collections can be usefully annotated in various ways, e.g., indicating their provenance, creation date, origin, etc.
Publishing Patterns

Collections of triples may be published to the web using a variety of syntaxes and mechanisms. It can be useful to partition data into a number of different documents, e.g. to simplify the publishing process or usefully present data to user agents. By annotating the documents to indicate their type we can usefully allow user agents to select specific collections that are more likely to contain information of interest. This can help target crawler behaviour or prioritise documents for de-referencing.

Using document types does not imply that a user agent can make assumptions about the structure or format of the data that will be retrievable. The document may contain information about any number of different resources, or use any RDF syntax.

Two well deployed examples of document typing in use today are RSS 1.0 and FOAF. RSS 1.0 is defined as both an XML and an RDF vocabulary and as such has a strict definition of document that aligns with its use in an XML context. FOAF however is an RDF vocabulary, but has still found it useful to define the notion of a foaf:PersonalProfileDocument which indicates that a document primarily describes a particular person (but may include additional data).

The Document Type pattern is most commonly used in conjunction with the See Also and Annotation patterns. It could also usefully be applied when referencing a Link Base, allowing a user agent to more easily discover Equivalence Links related to a specific resource(s).

Related

- Named Graphs
- See Also

Edit Trail

*How can people be encouraged to improve or fix up open data?*

Context

Community maintained or managed information sources often underpin a particular set of data. However by publishing the data in a machine-readable form the data may become dissociated from the community tools or forum that is used to create and maintain the data. Without a clear association from the data to the tools, users may be discouraged from using due to a perception of poor quality, when it is often within their means to fix up or improve the data.

Solution

Use the ex:editform property to associate a resource with a form, or other entry point into an application that can be used to correct the raw data.

Example(s)

Associating a dbpedia resource page with the wikipedia page from which its data is derived.

Discussion

The "Edit This Page" concept is well embedded into a number of community maintained websites, and particularly those based on a wiki platform. A convention has grown up on those sites that involves
providing a machine-readable link in the HTML page that allows a user agent to provide the user with a way to auto-discover links to the associated editing form.

The ex:editform property applies this concept to Linked Data resources. In this case the association is from the resource to editing form, allowing a more direct link from to be made from an information resource to a data management tool that can be used to edit its properties.

By associating the property directly with the resource, as opposed to the related page of which it is a primary topic, we aim to ensure that even the link remains even if only a fragment of the dataset is re-published.

Embedded Metadata

*How do we add structured data to an existing document or file?*

Context

There are two related aspects to this pattern

Firstly, many web sites that are using frameworks or content management systems that are either difficult to customize or are operated by organizations that have little or no in-house technical staff. How can these sites be updated, with a minimum of effort, to support publishing of structured data?

Secondly, documents or files may be shared and copied across the web. This can result in a broken connection between the document and it's metadata which might only be available from the source website. How can the metadata be made available to anyone discovering a copy or mirror of the document?

Solution

Embed the structured data directly in the document itself rather than, or in addition to, publishing that data separately.

The most common scenario here is instrumenting an existing webpage to add some "semantic markup" that will enable a client to extract data directly from the HTML source. This typically involves changing just the templates used ot generate the web pages. By changing a small number of resources, it becomes possible to quickly and easily publish data about a large number of resources.

A less common scenario involves embedding data within a different document format. Typically this relies on using an existing extension mechanism that has been defined for that format. A tool may then inspect the file directly to discover the metadata or a link to its location.

Example(s)

At the time of writing there are a number of competing proposals for embedding metadata in XHTML/HTML documents, including RDFa, microdata and microformats. RDFa can also be used to embedded metadata in other XML formats.

Options vary for embedding metadata in other formats, but Adobe XMP provides an option that can be used in a variety of formats.

Discussion

Embedding metadata into existing resources, rather than requiring changes to how content and data is published to the web is often much easier to achieve. At the time of writing there are a number of competing
approaches for embedding metadata in XHTML/HTML. These typically offer the same basic features but vary in how much they require specific markup extensions in the original document.

While it can be very easy to quickly instrument a website with embedded metadata, the requirement to introduce all data into the same template means that it can become awkward to manage the competing concerns: e.g. clean, well-structured markup for styling and presentation, versus rich, detailed, semantic markup for machines. For more complex use cases its may be better to simply publish the metadata separately, or embedded only a subset of the data with links to additional resources.

Related

- Primary Topic Autodiscovery
- See Also

Equivalence Links

How do we indicate that different URIs refer to the same resource or concept?

Context

Linked Data is published in a de-centralised way with multiple people and organisations collectively publishing data about the same resources. While it is a goal to reuse identifiers wherever possible this may not always be achievable. If data has been published in this way, how can links be built between datasets in order to identify that distinct URIs refer to the same resource?

Solution

Use owl:sameAs, skos:exactMatch, or similar specialised relation, to indicate that two URIs are equivalent.

Example(s)

```
#Statement from Data Set 1
ex:bob a foaf:Person.

#Statement from Data Set 2, with equivalence
ex:otherBob a foaf:Person.
ex:otherBob owl:sameAs ex:bob.
```

Discussion

Distributed publishing is a fact of life on the web. Semantic web technologies have built-in support for handling distributed publishing of both data and schemas by standardising specific relations for bringing together disparate datasources. The most important of these facilities is the owl:sameAs relation, which indicates that two URIs are equivalent. According to the semantics defined in the OWL standard, the two URIs are synonyms of each other and all statements made about one of those URIs is also considered to be true of the other.
With support from a reasoner the `owl:sameAs` relation allows a semantic web application to query and interact with a single view across any number of datasets. This allows data to be integrated, without requiring any merging or re-processing of the data. It also avoids the need for up-front convergence on standard identifier schemes.

However `owl:sameAs` is only one possible equivalence relation that could be stated between resources. Other vocabularies may choose to define additional relationships that have less strong semantics associated with them. For example SKOS defines two additional properties, `skos:closeMatch` and `skos:exactMatch`, that can be used to state equivalences between concepts in a thesaurus. The relations allow for more fuzzy notions of equivalence and have weaker semantics: `skos:exactMatch` declares two concepts to be the same but doesn't imply that all statements about one concept are also true of another.

OWL also defines some additional specialised equivalence relations for relating together classes (`owl:equivalentClass`) and properties (`owl:equivalentProperty`). These should be used instead of `owl:sameAs` when relating together terms in an ontology.

**Related**

- Proxy URIs
- Link Base

**Link Base**

_How can outbound links from a dataset be managed separately from the core data?_

**Context**

When publishing Linked Data it is common to undertake the inter-linking of newly published data with existing sources as a final step in the publishing process. Linking strategies range from manual methods through to automated link discovery. The volume of out-bound links may vary over time, e.g. as new links are discovered to existing sources, or new links are made to other new datasets. Over time, due to "semantic drift", it may be necessary to remove links to some external resource.

The linking data within a dataset may therefore have different quality assurances associated with it, as well as a different rate of change to the core dataset.

**Solution**

Partition the data into two datasets: the core data and the linking data. Publish linking data as a separate set of documents. Use See Also links to tie the data together.

**Example(s)**

#Core Data
```
<http://www.example.org/places/Paris> a ex:Place;
   skos:prefLabel "Paris";
   rdfs:seeAlso <http://www.example.org/links/Paris>.
```
#Linking Data

`<http://www.example.org/links/Paris>

**Discussion**

Partitioning links from core data has a number of advantages in terms of both how it is published and how it is consumed.

For the publisher, partitioning links allows the linking data to be revised and improved separately to the publication of the core reference data. This is particularly useful where the linking data is generated automatically using heuristics. The quality of the generated links may need to be refined and iterated over time. As new datasets are published, the linking data can get updated to include additional relationships. Managing the typically smaller subset of links distinctly from the core data, e.g. in a separate triple store, allows more flexibility in managing the data.

For a consumer, the partitioning of links into a separate dataset allows more choice in what data is to be consumed. Because the quality characteristics of linking data may vary considerably from that of the core data, the ability to selectively consume only the core data is an important one. If the linking data is presented as part of the core dataset, then this requires the consumer to filter the data to remove the unneeded triples.

**Related**

- Equivalence Links

**Materialize Inferences**

*How can data be published for use by clients with limited reasoning capabilities?*

**Context**

Linked Data can be consumed by a wide variety of different client applications and libraries. Not all of these will have ready access to an RDFS or OWL reasoner, e.g. Javascript libraries running within a browser or mobile devices with limited processing power. How can a publisher provide access to data which can be inferred from the triples they are publishing?

**Solution**

Publish both the original and some inferred (materialized) triples within the Linked Data.

**Discussion**

Reasoners are not as widely deployed as client libraries for accessing RDF. Even as deployment spreads there will typically be processing or performance constraints that may limit the ability for a consuming application to perform reasoning over some retrieved data. By also publishing materialized triples a publisher can better support clients in consuming their data.

Most commonly materialization of the inferred triples would happen through application of a reasoner to the publishers data. However a limited amount of materialized data can easily be included in Linked Data.
Publishing Patterns

views through simple static publishing of the extra relations. E.g. by adding extra "redundant" statements in a template.

There are a range of useful relationships that could be included when implementing this pattern:

- Typing, based on range and domains of properties and/or derived classes
- Super properties, based on property extensions
- Transitive relationships, e.g. skos:broadr and skos:narrower relations
- Inverse properties

Materialization may also be targeted in some way, e.g. to address specific application needs, rather than publish the full set of inferred relations. The specific materializations chosen can be based on expectations of common uses for the data. For example the publisher of a SKOS vocabulary may publish transitive relations between SKOS concepts, but opt not to include additional properties (e.g. that every skos:prefLabel is also an rdfs:label)

A reasonable rule of thumb approach to materializations would be to always include additional type or property relations whenever that would help ground the data in more commonly used vocabularies. Inverse and transitive relationships provide extra navigation options for Linked Data clients so are also worth considering.

The downside to publishing of materialized triples is that there is no way for the consuming system to differentiate between the original and the inferred data. This limits the ability for the client to access only the raw data, e.g. in order to apply some local inferencing rules. This is an important consideration as publishers and consumers may have very different requirements. Clearly materializing triples also places additional burdens on the publisher.

An alternative approach is to publish the materialized data in some other way, e.g. in a separate document(s) referenced by a See Also link.

Further Reading

- Creating Linked Data - Part V: Finishing Touches [http://www.jenitennison.com/blog/node/139] (see section on Derivable Data)

Primary Topic Autodiscovery

*How can people identify the principal subject of a given web page?*

Context

Often a web page is concerned with a particular physical object or other resource. To assist discovery and aggregation of pages about particular topics the publisher wants to indicate the URI of this resource.

Solution

When publishing a web page include a link element in the head of the web page pointing to the URI of the page's primary topic. Use a rel attribute value of "primarytopic".

```html
<link rel="primarytopic" href="http://dbpedia.org/resource/London"/>
```
Example(s)

Associating a wikipedia page with the equivalent dbpedia resource.

Discussion

Many pages on the Web are explicitly about a single subject. Examples include Amazon product pages, Wikipedia entries, blogs and company home pages. Without an explicit link, content aggregators must resort to heuristic inference of the topic which is prone to classification error. Often the original publisher knows the specific topic and would like to provide this as a hint to aggregators and other content consumers.

Even when the page is about several topics there can be a single primary topic that can be linked to directly.

Related

• Autodiscovery

Progressive Enrichment

*How can the quality of data or a data model be improved over time?*

Context

At the time when a dataset is first published the initial data may be incomplete, e.g. because data from additional systems has not yet been published, or the initial dataset is a place-holder that is to be later annotated with additional data. Data models are also likely to evolve over time, e.g. to refine a model following usage experience or to converge on standard terms.

Solution

As the need arises, update a dataset to include additional annotations for existing or new resources.

Discussion

A key benefit of the semi-structured nature of RDF is the ability to easily merge new statements into an existing dataset. The new statements may be about entirely new resources or include additional facts about existing resources. There is no need to fully define a schema, or even fully populate a data model, up front. Data can be published and then refined and improved over time.

Progressive Enhancement is essentially a variant of the Annotation pattern within a single dataset. Whereas the Annotation pattern describes an approach to distributed publishing of data about a set of resources, Progressive Enhancement confines this to a particular dataset allowing the depth of detail or quality of the modelling to improve over time.

A common use of this pattern in Linked Data publishing is to update a dataset with additional Equivalence Links.

Progressive Enrichment is a key aspect of the Blackboard application pattern.

Related

• Annotation
• Equivalence Links

See Also

*How can RDF documents be linked together to allow crawlers and user agents to navigate between them?*

Context

Linked Data is typically discoverable by de-referencing URIs. Starting with a single URI a user agent can find more data by discovering other URIs returned by progressively retrieving descriptions of resources referred to in a dataset. However in some cases it is useful to provide additional links to other resources or documents. These links are not semantic relations per se, just hypertext links to other sources of RDF.

Solution

Use the `rdfs:seeAlso` property to link to additional RDF documents.

Example(s)

The Linked Data published by the BBC Wildlife Finder application includes data about ecozones. The data about an individual ecozone, e.g. the Nearctic Ecozone [http://www.bbc.co.uk/nature/ecozones/Nearctic_ecozone.rdf] refers to the habitats it contains and the species that live in that ecozone. A semantic web agent can therefore begin traversing the graph to find more related data. The RDF document returned from that URI also includes a seeAlso relationship to another document that lists all ecozones.

Discussion

The `rdfs:seeAlso` relationship is intended to support some hypertext links between RDF documents on the web. There are no explicit semantics for the property other than that a user agent might expect to find additional, relevant RDF data about a resource at the indicated location. Using this relationship allows documents to be linked together without requiring semantic relations to be specified between resources where none exists.

By ensuring that data from a Linked Data site is robustly linked together, it helps semantic web crawlers and user agents to traverse the site to find all relevant material. The `rdfs:seeAlso` relation is therefore well-suited for publishing simple directories of links for a crawler to follow.

The relation can also be used to refer to other documents on the web, e.g. published by third-parties, that may contain additional useful Annotation data.

Related

• Annotation

Unpublish

*How do we temporarily or permanently remove some Linked Data from the web?*

Context

It is sometimes necessary to remove a Linked Data set from the web, either in whole or in part. A dataset might be published by an organisation who can no longer commit to its long term availability. Or a dataset
might be transferred to a new authority. This applies to scenarios where a third-party has done a proof-of-concept conversion of a dataset that is later replaced by an official version.

In practical terms a dataset might also be temporarily unavailable for any number of technical reasons.

How can the temporary or permanent removal of some data be communicated? And, in cases where it has been replaced or superceded, how can the new authoritative copy be referenced.

**Solution**

Use an appropriate HTTP status code to indicate the temporary or permanent removal of a resource, or its migration to a new location.

Where a resource has moved to a new location, publish Equivalence Links between the old and the new resources.

**Example(s)**

A dataset has been published by a developer illustrating the benefits of a Linked Data approach to data publishing. The developer has used URIs based on a domain of http://demo.example.net. At a later date the original owner of the data decides to embrace Linked Data publishing. The new dataset will be published at http://authority.example.org.

The developer therefore reconfigures his web server to redirect all URIs for http://demo.example.net to return a 301 redirect to the new domain. Consuming applications are then able to determine that the data has been permanently moved to a new location.

The developer also creates a data dump that contains a series of RDF statements that indicate that all of the resources originally available from http://demo.example.net are owl:sameAs the new official URIs.

**Discussion**

Movement or removal of web resources is not specific to Linked Data, and so HTTP offers several status codes that are applicable to the circumstances described in this pattern. Using the correct HTTP status code is important to ensure that clients can differentiate between the different scenarios. An HTTP status code of 503 indicates that a resource is temporarily unavailable; 410 that a resource has been deleted; and 301 that a resource has been moved to a new location. Returning 404 for a resource that is only temporarily unavailable, or has been moved or deleted is bad practice.

Where data has been replaced, e.g. new URIs have been minted either at the same authority or a new one, then publishing RDF assertions that relate the two URIs together is also useful. An owl:sameAs statement will communicate that two URIs are equivalent and will ensure that any historical annotations associated with the URI can be united with any newly published data.

Lastly, in case of complete removal of a dataset, it is important to consider archiving scenarios. If licensing permits, then data publishers should provide a data dump of a complete dataset. Doing so will mean that consumers, or intermediary services, can host local caches of the data to support continued URI resolution (e.g. via a URI Resolver). This mitigates impacts on downstream consumers.

**Related**

- Equivalence Links
• URI Resolver
Chapter 5. Data Management Patterns

Abstract

While the statements in an RDF dataset describe a direct graph of connections between resources, the collection of triples itself has no structure: it is just a set of RDF statements. This lack of structure is not a problem for many simple RDF applications; the application code and behaviour is focused on exploring the connections in the graph. But for more complex systems that involve integrating data from many different sources it becomes useful to be able to partition a graph into a collection of smaller sub-graphs.

One reason for partitioning of the graph, is to support data extraction. Creating a useful view over one or more resources in a graph, e.g. to drive a user interface. There are a number of different partitioning mechanisms that can be used and these are covered in the Bounded Description pattern described in the next chapter.

A very important reason for wanting to partition a graph is to make data management simpler. By partitioning a graph according to its source or the kinds of statements it contains we can make it easier to organise and update a dataset. Managing smaller graphs gives more affordance to the data, allowing entire collections of statements to be manipulated more easily.

The means by which this affordance is created is by extending the core triple model of RDF to include an extra identifier. This allows us to identify collections of RDF triples, known as Named Graphs. The patterns captured in this chapter describe different approaches for managing RDF data using Named Graphs. The patterns cover different approaches for deciding on the scope of individual graphs, as well as how to annotate individual graphs, as well as ultimately re-assembling graphs back into a useful whole.

It should be apparent that Named Graphs is essentially a document-oriented approach to managing RDF data. Each document contains a collection of RDF statements. This means that we can benefit from thinking about good document design when determining the scope of each graph, as well as more general document management practices in deciding how to organise our data.

The beauty of the RDF model is that it is trivial to manage a triple store as a collection of documents (graphs) whilst still driving application logic from the overall web of connections described by the statements contained in those documents. An XML database might also offer facilities for managing collections of XML documents, but there is no standard way in which the content of those documents can be viewed or manipulated. In contrast the data merging model described by RDF provides a principled way to merge data across documents ((Union Graph).

This flexibility provides some powerful data management options for RDF applications.

Graph Annotation

How can we capture some metadata about a collection of triples?

Context

There are a number of scenarios where it is useful to capture some metadata about a collection of statements in a triplestore. For example we may want to capture:

• publication metadata, such as the date that the triples were asserted or last updated

• provenance metadata, e.g. who asserted those triples, or how they were generated
Data Management Patterns

- access control data, e.g. which user(s) or role(s) can access those triples

The Named Graph pattern allows us to identify a set of triples, via a URI. But how do we then capture information about that graph?

**Solution**

Treat the Named Graph like any other resource in your dataset and make additional RDF statements about the graph itself. Those additional statements can themselves be stored in a further named graph.

**Example(s)**

A triple store contains a Named Graph that is used to store the results of transforming a local database into RDF. The Named Graph has been given the identifier of http://app.example.org/graphs/my-database to label the triples resulting from that conversion process. As the source is a live, regularly updated database, it is useful to know when the RDF conversion was last executed and the data stored. It is also useful to know which version of the conversion software was used, to track potential bugs. This additional metadata could be captured as follows:

```r
@prefix ex: <http://www.example.org/> .

#Named graph containing results of database conversion
<http://app.example.org/graphs/my-database> {  
  ...triples from conversion...
}

#Metadata graph
<http://app.example.org/graphs> {  
  #Description of a named graph in this dataset
  <http://app.example.org/graphs/my-database> foaf:generatorAgent <http://app.example.org/converter/0.0.5>.
  ...
}
```

In the above example there are two graphs in the dataset: the graph containing the data from the conversion and a second graph containing metadata about the first. An application that needed to identify the date that some triples were asserted can easily do this by querying the metadata graph.

**Discussion**

The majority of Named Graph uses cases require some additional context to be captured about the set of triples labelled with a graph URI. Describing named graphs in RDF, by using the graph URI as the subject of additional RDF statements, provides a simple way to capture additional metadata relevant to an application.

As shown in the above example, a specific "well-known" named graph in the dataset can be designated as the location in which these extra statements are recorded. However, it is also possible to use multiple named graphs. In this case, we might have one graph for the data and one "parallel" graph that captures the context. For example, if we could adjust the above example as follows:
@prefix ex: <http://www.example.org/> .

#Named graph containing results of database conversion
<http://app.example.org/graphs/my-database> {  
    ...triples from conversion...
}

#"Parallel" metadata graph
<http://app.example.org/metadata-graphs/my-database> {  
    #Description of named graph in this dataset
    <http://app.example.org/graphs/my-database> foaf:generatorAgent <http://app.example.org/converter/0.0.5>.}

...other named graphs

Using a pair of named graphs per source can quickly lead to a very large number of graphs in a dataset. Some triple stores may not be optimized to deal with a very large number of graphs. However the approach does benefit from flexibility of adding and removing both source graphs and their metadata.

Care should be taken when choosing graph URIs. If an application uses the source document URL of some RDF as the named graph URI then this can lead to confusing statements as illustrated below:

@prefix ex: <http://www.example.org/> .

#Named graph containing results of HTTP crawl
<http://example.org/person/joe.rdf> {  
    #statements contained in the document, about itself
    #date the FOAF document was created
    <http://example.org/person/joe.rdf> dct:created "2010-06-01".

    <http://example.org/person/joe> foaf:name "Joe Bloggs".
}

#Metadata graph, describing results of crawl
<http://app.example.org/graphs/> {  
    #date the named graph was created
    <http://example.org/person/joe.rdf> dct:created "2012-05-28".
}

In the above example there are two dct:created properties associated with the resource http://example.org/person/joe.rdf. That document is a FOAF description which describes its primary topic and the date it was generated. The second date was added by a hypothetical web crawler that captured the date it stored that information in a named graph. It is possible for a SPARQL query applied to the union graph for this dataset to return conflicting information, suggesting that the use of the source URL as a graph identifier is a poor choice. A better alternative would be to add some indirection:

@prefix ex: <http://www.example.org/> .

#Named graph containing results of HTTP crawl
In the revised example a new URI is associated with the results of the web crawl. The web crawler then uses this URI to record the required metadata. By adding a dct:source property (or similar) it is still possible to identify which named graph was derived from which source URL. As can be seen a Patterned URI is used to generate the graph identifier, giving a predictable structure to the store.

Related

- Named Graph
- Graph Per Resource
- Graph Per Source

Further Reading

- Named Graph (Wikipedia)

Graph Per Aspect

_How can we avoid contention around updates to a single graph when applying the Graph Per Resource pattern?_

Context

For some applications the entire description of a resource might be maintained by a single authority, e.g. the data might all derive from a single data conversion or be managed by a single editing interface. However in some applications data about a single resource might be contributed in different ways. One example might be a VoID description for a dataset. A dataset description may consist of a mixture of hand-authored information -- e.g. a title, description, and example resources -- plus some statistics derived from the dataset itself, e.g. size and class partitions. An administrator might update the descriptive aspects while the rest is updated asynchronously by a background application that analyses the dataset.

Multiple applications writing to the same graph could lead to contention for system resources or the need to implement complex locking behaviour.
Solution

Apply a combination of the Graph Per Resource and Graph Per Source patterns and factor out the different aspects of a resources description into separate graphs. Use a Union Graph to collate the different aspects of the description of a resource into a single view.

Example(s)

A content management application stores information about articles. This includes descriptive metadata about the articles as well as pointers to the content. Content metadata will be manually managed by users. In the background two additional processes will be carrying out additional tasks. One will be retrieving the content of the article to perform text mining, resulting in machine-tagging of subjects in the article. The second will be harvesting related links from the rest of the system and the web. The "aspect graphs" are created: one for the core metadata, one for the tags and one for the links:

```turtle
#core description of a resource; provided by user
<http://data.example.org/graphs/core/document/1> { 
  <http://example.org/document/1> dct:title "Bath in the Summertime".
}
#tags; maintained by process 1.
<http://data.example.org/graphs/tags/document/1> { 
  <http://example.org/document/1> dc:subject "Bath".
  <http://example.org/document/1> dc:subject "Travel".
}
#related links; maintained by process 2.
<http://data.example.org/graphs/links/document/1> { 
}
#System metadata graph, listing topic of each graph
<http://data.example.org/graphs> { 
}
```

As the above example illustrates, graph URIs for the different aspects of a resources description can by generated by using Patterned URIs. A fourth graph, covering system-wide metadata is also maintained. This graph lists the foaf:primaryTopic of each graph, allowing applications to discover which graphs relate to a specific resource.

An application consuming this data could rely on either a system default Union Graph to provide a complete view of a resource. Partial views might address individual named graphs. Using a CONSTRUCT query it is also possible to construct a view of a resource using just those graphs referenced in the system metadata graph:

```
CONSTRUCT { ?s ?p ?o. }
WHERE {
  <http://data.example.org/graphs> { 
  }
}
```
Discussion

Named graphs provide flexibility in how to organise an RDF data store. In some cases storage is oriented towards the sources of data, in others around individual resources. The Graph Per Aspect pattern provides a combination of those features that allows for very fine-grained graph management. The description of each resource is divided over a number of graphs, each of which is contributed to the system by a different source or application component.

As with the other named graph patterns reliance is made on the Union Graph pattern to bring together the description of a resource into a single consistent view.

Separating out aspects of resource description into different graphs also provides a way to shard a dataset. Different aspects might be stored in different triple stores across a network. These are then brought together in the application for building a user interface. With knowledge of how graphs are partitioned across the network, as well as which graphs contain which statements, an application can use Parallel Retrieval to synthesis a local working copy of a resource's description. This aggregation could happen within a server component or on the client-side.

The small, focused graphs created by use of this pattern and, more generally, by the Graph Per Resource pattern are very amenable for delivery to mobile & web clients for local processing. By separating out the different aspects of a resource into graphs that are likely to change with differing frequencies, caching can be made much more efficient.

Related

- Named Graph
- Graph Annotation
- Graph Per Source
- Graph Per Aspect

Further Reading

- Keep Your Triples Together: Modeling a RESTful, Layered Linked Data Store

Graph Per Resource

*How can we organise a triple store in order to make it easy to manage the statements about an individual resource?*

Context

Web applications typically offer forms for editing the description of an individual resource, e.g. the title, description and tags that apply to a specific photo. RESTful APIs typically support replacing the description of a resource using a PUT operation. In both cases it would be useful to be able to serialize all
of the statements relating to a given resource, including any edits, and directly replace the relevant triples in a triple store backing the application or API.

**Solution**

Store the description of each resource in a separate Named Graph using a graph URI derived from the resource URI as the graph URI. When the resource is updated, simply replace the contents of that graph with the latest state of the resource.

**Example(s)**

A user is editing the description of http://example.org/picture/1. The application delivers a web form to the authorised user which presents the current description of the resource in an interactive editing form. When the client submits the changes back to the server, it does so by serializing the state of the form as RDF which is then processed by the server.

On receiving the update from the client, the server-side code computes the graph URI for the resource being edited. For this application graph URIs are derived from resource URIs by a simple rewrite. For example the data for http://example.org/picture/1 is stored in http://data.example.org/graphs/picture/1.

To apply the update the server-side code simply determines the appropriate graph URI and then stores the data in a SPARQL 1.1. Graph Store Protocol enabled store using a simple PUT operation.

**Discussion**

Partitioning the triple store by resource rather than by source provides an easy way to quickly access the description of an individual resource without having to use a SPARQL query. This is particularly true if the graph URI for a resource can be derived using a simple algorithm, e.g. rewriting to a different base URI, pre-pending a known prefix, etc.

The SPARQL 1.1. Graph Store Protocol provides a way for code to find and update the description of a resource without needing further knowledge of the data stored about any specific resource. This allows server-side code to remain relatively generic: it doesn't need to know details about what kinds of data is captured about individual resources, it just needs to know where to persist the data its given.

When applying this pattern it is common to store a bounded description of the resource in each graph ensuring that each graph has a clear scope.

While this pattern makes it easy to manage a bounded description for a resource, including its relationships to other resource, it doesn't help with managing references to the resource elsewhere in the store. E.g. if a resource is deleted (by removing its graph), there may still be statements relating to that resource elsewhere in the graph. Additional SPARQL Update operations, for example, would be needed to remove these statements. Depending on the complexity of the graph structure and the needs of the application these overheads might make this pattern unsuitable.

In circumstances where additional context is required, e.g. in-bound relations, for an application view, a fall-back to SPARQL queries over a Union Graph of the store can provide additional flexibility. However for the simplest cases of serving Linked Data pages, this pattern makes for very simple and light-weight application code.

For very simple use cases the graph URI of the graph holding the description of the resource could in fact be the resource URI. However in practice this can limit system evolution. As explained in the Graph Annotation pattern, the ability to store additional graph metadata can be useful. And in that case a separate graph URI is essential to avoid potential confusion around the scope of individual statements.
In some applications several different processes might contribute to the creation of a description of a resource. Using a single graph per resource might therefore involve contention around several processes. The Graph Per Aspect pattern allows this contention to be removed by using multiple graphs per resource.

Related

- Named Graph
- Graph Annotation
- Graph Per Source
- Graph Per Aspect

Further Reading

- Named Graph (Wikipedia)
- Managing RDF using Named Graphs

Graph Per Source

_How can we track the source of some triples in an RDF dataset?_

Context

A common application requirement is to create a local cache of RDF data which originates from other sources. For example an application might crawl or harvest data from the Linked Data cloud and store it local triple store. Data from relational databases or other sources might also be regularly converted into RDF and added to a triple store.

While the applications directly consuming this data may not need any knowledge of their origin when querying the dataset, the applications doing the data management activities (e.g. crawling or data conversions) will need to be able to clearly identify which triples came from which source dataset or URL.

Solution

Use a separate named graph for each data source. For the named graph URI either use a well-known URI for each data source, e.g. the URI of the dataset, or simply the URL from which the data was retrieved.

Example(s)

An application is harvesting RDF documents from the web. Upon discovering the URL http://www.example.org/person.rdf the application does a GET request to retrieve the document. The RDF statements found in that document are then stored in the applications triple store in a new named graph with the source URI as the graph identifier:

```rdfs
#Named graph URI is source document
<http://www.example.org/person.rdf> {  
  #Triples from source document
  <http://www.example.org/person/joe> foaf:name "Joe Bloggs".
}'''
```
The harvesting application can easily determine whether it has already harvested a URL by checking to see whether a named graph with that URI exists in the store. The next time that the application retrieves the content of the URL, e.g. as part of a cache refresh, the contents of just that graph can be replaced.

**Discussion**

The URI that labels a named graph can be used in several different ways. One very common approach is to use the label to hold the URL from which a set of triples was retrieved. By using the graph URI to identify the source of the data it becomes possible to track the provenance of the data within the store. This can simplify common dataset construction scenarios, e.g. mirroring of data harvested from the web.

Checking for the presence of a graph within a store can be achieved using a simple assertion query:

```
ASK WHERE {
    <http://www.example.org/person.rdf> { ?s ?p ?o. }
}
```

The query will return true if there are any statements associated with a graph with the specified identifier.

An application may need to be present to an end-user the list of locations from which some RDF statements have been found. This too can be achieved using a simple SPARQL query:

```
SELECT ?graph WHERE {
    ?graph {
        <http://www.example.org/person/joe> foaf:name "Joe Bloggs".
    }
}
```

In the above example the query will return the URIs of all graphs that contain the specified RDF triple; "which sources say that http://www.example.org/person/joe has a foaf:name of "Joe Bloggs"."

It is often important to capture some additional information about the source of the URL. For example a web crawler might want to record the retrieval data and HTTP headers associated with the original web request. This would allow for more efficient cache maintenance. The Graph Annotation pattern describes this in more detail.

It is worth noting that in some cases a direct application of this pattern can make graph annotation more awkward: by using the source URL as the graph URI we can no longer distinguish between statements made about the graph and statements made about the source document. E.g. the date of creation of the graph and the date of creation of the document. This may be an issue for some application scenarios, although for others (e.g. simple aggregation) this may not be an problem.

As described in the Graph Annotation pattern the solution to this issue is to use a different identifier for the graph, e.g. a Patterned URI derived from the source URL. The source URL of the graph can then be captured as an annotation.

**Related**

- Named Graph
- Graph Annotation
- Graph Per Resource
Further Reading

- Named Graph (Wikipedia)

Named Graph

*How can we identify a useful sub-set of RDF triples within a triple store?*

Context

A triplestore is a set of RDF triples. Upon loading some RDF data into such a collection we lose the ability to be able to identify a sub-set of RDF triples, e.g. to identify their source of origin. We also lose knowledge of whether a single triple was asserted once or multiple times, e.g. by separate distinct sources. While it is possible to extract triples based on particular graph patterns, e.g. all triples with the same subject, predicate, or object there is no way to recover the original context.

Solution

Use URIs to identify collections of statements. Where necessary associate a triple with this URI to create a *quad*

By assigning a URI to a set of triples and my retaining that additional identifier in the store (a *quad store*), we can either treat the whole collection as a single set of triples, or work with an subset based on its graph identifier.

Example(s)

The following example shows an RDF graph using the TRiG notation (Turtle plus graphs)

```turtle
@prefix ex: <http://www.example.org/> .

#Named graph in TRiG
<http://www.example.org/graph/1> {   
  <http://www.example.org/document/7> rdfs:label "Example".
}
```

The following example shows an RDF graph using the NQuads notation (N-Triples plus graphs)

```turtle
```

In both cases the graphs contain a single RDF triple

Discussion

The idea of extending the triple into a quad, through the addition of an additional URI has been around for many years. The Many early triple stores supported the notion of a quad, with the majority now supporting named graphs as a key feature. SPARQL 1.0 was the first semantic web specification to directly reference
the concept, allowing queries to be addressed to an individual named graph or a collection of graphs. Syntaxes for serialising named graphs also exist, allowing quads to be exchanged between systems. The most commonly used quad syntaxes are TRiG (a derivative of Turtle) and NQuads (a derivative of NTriples.

By extending the core RDF model from a triple to a quad, Named graphs provide a useful extra degree of freedom when managing an RDF dataset. A useful analogy when thinking about a quad store is that of a collection of documents: each named graph is a separate "document" in the database that can be manipulated independently from any others. However the document identifier (the graph URI) can be ignored when working with data, relying on RDF graph merging rules to provide a view across all documents. A graph store therefore offers a useful way to manage sets of statements without losing the ability to easily merge data from across sources.

Graphs have been usefully applied to solving a number of different data management problems in RDF applications. Some recorded uses for Named Graphs include:

• *Tracking provenance of RDF data* — here the extra URI is used to track the source of the data; especially useful for web crawling scenarios

• *Replication of RDF graphs* — triples are grouped into sets, labelled by a URI, that may then be separately exchanged and replicated

• *Managing RDF datasets* — here the set of triples may be an entire RDF dataset, e.g. all of dbpedia, or all of musicbrainz, making it easier to identify and query subsets within an aggregation

• *Versioning* — the URI identifies a set of triples, and that URI may be separately described, e.g. to capture the creation & modification dates of the triples in that set, who performed the change, etc.

• *Access Control* — by identifying sets of triples we can then record access control related metadata

The lack of standardisation about what the graph URI associated with a named graph identifies means that different applications have used it for different purposes. For example one application might simply use the URI as a local identifier for a set of triples, whereas another might associate extra semantics with the URI, e.g. using it to hold the URL from which the data was originally retrieved.

The simplest way to think about the graph URI is as a simple identifier or label that can be associated with some triples. The fact that the identifier is a URI has some added benefits. For example we could then capture RDF statements to describe the graph. This has been applied as an alternative to reification for handling versioning and provenance in datasets, but graph annotation is useful in a number of scenarios.

**Related**

• Reified Statement
• Graph Annotation

**Union Graph**

*How can we generate a simple view over the statements in a collection of named graphs?*

**Context**

The Named Graph pattern describes an approach for managing RDF statements as a collection of graphs. The Graph Annotation pattern illustrates how to capture information about these graphs. However for many use cases, a consuming application may need to ignore the graph structure of a dataset and instead
work on a single consistent view of the data. In other circumstances we may only want to query a certain set of graphs (e.g. just the data, or just the annotations). So how do we gain the benefits of named graphs without having to teach all applications about the internal structure of our triple store?

Solution

Write queries against a "union graph" that provides a simple view over a collection of named graphs by relying on RDFs data merging rules.

Example(s)

See below

Discussion

Named graphs are a useful way to structure a store to simplify the compilation and maintenance of a dataset. But we might consider that the graph structure of a dataset -- i.e. how it is composed of a number of named graphs -- to be an implementation detail that needn't concern applications built on that data.

The RDF data model provides rules for merging of RDF statements into a set. Those rules can be applied to create a simple synthetic view over a collection of named graphs that ignores the graph structure completely. A union graph is the RDF merge of one or more named graphs.

Most RDF triple stores offer the ability to create or use union graphs in some form. SPARQL also provides some options for creating additional "union graphs" (otherwise known as "synthetic" or "view" graphs).

We can think of a quad store as a set of named graphs, each of which has an identifier. All triples are therefore associated with a specific named graph. However when applying some pattern matching, e.g. in a query, or inferencing rules, we can safely ignore the graph URI component of the quad, this result in a simple triple based view of the data. This is a simple union over all the graphs. Some stores allow this union to be referenced in a query or application code, in order to allow graph-agnostic processing.

In SPARQL a dataset consists of a default graph, which doesn't have an identifier, plus zero or more named graphs which each have a URI. Triple stores that are geared towards SPARQL may similarly offer a default graph in which triples can be stored and the ability to store data in additional named graphs. TDB (part of Apache Jena) provides such a facility. Triples may be added to a dataset and these are then stored in the default, unnamed graph. In a SPARQL query, triples patterns are matched only against this graph. However TDB also offers a way to address the union graph of the store: i.e a view over all of the named graphs in the store (plus the default). TDB may also be configured to not allow updates to the default graph. In this case the default graph is automatically constructed from the union graph.

Other triple stores do not support an default unnamed graph, requiring all triples to be associated with some graph. In this case when executing a SPARQL query the default graph will be selected based on some system wide default (e.g. the union of all graphs) or by the query itself using a FROM clause.

The SPARQL FROM clause provide another way to define custom union graphs. The FROM clause is used to identify the default graph for a query. The most typical use is to identify a single RDF graph. However if multiple FROM clauses are specified in a query then the contents of those graphs are merged (typically in-memory) to provide a union graph that will form the default graph for the query. This feature of SPARQL can therefore provide another way to assemble a useful graph-agnostic view of a dataset.

The options here are on the surface confusing. But they offer some useful further options for managing and querying datasets. The important part is to understand that some options are provided by the underlying storage, whereas others are a function of the SPARQL query language:
• a triple store may manage data either as a collection of named graphs, or directly as a SPARQL dataset, i.e. a default graph plus zero or more named graphs. In the latter case triples can be added/removed from the default graph

• a triple store may provide a view over its entire contents, regardless of the partitioning into graphs. This is the most common form of union graph. It is also likely to be efficient for pattern matching, etc. This union graph may be in addition to whatever basic storage options the store provides

• a triple store may offer options for how a sparql dataset is created for a given query, or it may enforce a specific structure. E.g. a store may enforce that the default graph is a specific stored graph, or that it is a union graph providing a view of all of its contents

• a triple store may offer some additional flexibility for a sparql query to define its dataset including specifying a single graph as the default, the store-wide union graph, or a query specific union graph which is populated by merging together a number of named graphs identified in the FROM clause of the query

Not all triple stores offer this kind of flexibility but many offer at least some options for querying the whole store as a union graph.

There is scope here for further innovation on the part of store implementors, e.g. to offer additional ways to either statically or dynamically create union graphs over a store. For example an application may want to identifier the most recently updated graphs; graphs with a particular kind of content; public and "private" graphs, etc.

**Related**

• Named Graph

• Graph Annotation

**Further Reading**

• Managing RDF using Named Graphs

• TDB Datasets

• TDB Dynamic Datasets

• SPARQL 1.1 Specifying RDF Datasets
Chapter 6. Application Patterns

Abstract

Application architecture has always been a rich source of design patterns. Much of the design pattern literature covers useful architectural and code design patterns that can lead to the creation of more maintainable and evolvable software.

We are still at the early stages of exploring how best to create Linked Data applications. Indeed there is still much debate about what constitutes a Linked Data application at all. Is it any application that uses RDF, perhaps entirely based on a local triple store? Or must it be an application that is capable of continually discovering new information from across the web of data?

We still have much to learn about how to create applications that are truly flexible enough to process and display a wide variety of different data types. This covers everything from software architecture, design and user experience. Over time we might expect to see more research and development of Linked Data browsers. Or, perhaps web browsers will simply become more data aware and develop ways to help users make more of the data that is increasingly embedded in or linked from web pages.

Regardless of the type of applications we are constructing, there are a number of ways that specific features of RDF, SPARQL, or HTTP accessible Linked Data can be exploited to create flexible software architecture or simply useful behaviour. This chapter captures a variety of design patterns that relate to a number of different aspects of application development.

The existing patterns literature is at our disposal for helping to create Linked Data applications, but which features of semantic web technology can help us to better solve problems or meet requirements?

Assertion Query

How can a dataset be tested for known patterns?

Context

There are a number of circumstances in which it is useful to test for patterns in a dataset. The most common is likely to be discovery of whether there is data available about a specific resource or from a specific vocabulary. Another very common use of pattern detection is to validate a dataset to check that it conforms to a specific structure. The latter is very common when testing the data generated by an application, e.g. during development or ongoing maintenance, or to check data received from third-party systems.

Solution

Use a SPARQL ASK or CONSTRUCT query to probe the dataset for known patterns

Example(s)

The following will return true if the specified pattern is found:

#Is there any data about a specific resource?
ASK WHERE {
}

Discussion

As described in the Transformation Query pattern, SPARQL provides a declarative syntax for certain kinds of operations on RDF graphs. The SPARQL ASK query form is intentionally provided to support making assertions against an RDF dataset. It is therefore useful in the role of testing the output of an application (e.g. for unit testing) or acceptance testing incoming data received from a third-party.

In some cases it is useful to run a number of queries against a dataset and generate suitable warning messages if any of the assertions succeed. Instead of using an ASK query, a CONSTRUCT query can be used instead, allowing a simple message to be generated as output. These messages can then be used to, e.g. produce a test report from a dataset:

CONSTRUCT {
  _:msg ex:message "Every person should have a name".
} WHERE {
  ?person a foaf:Person;
  OPTIONAL {
    ?person foaf:name ?name.
  }
  FILTER (!bound(?name))
}

Related

• Transformation Query

Further Reading

• The Schemarama [http://isegserv.itd.rl.ac.uk/schemarama/] testing framework uses the CONSTRUCT variant of this pattern.

Blackboard

*How can the task of compiling or constructing a dataset be divided up into smaller tasks?*

Context

Applications consuming Linked Data commonly need to compile a dataset by collecting data from a number of distributed sources, e.g. other Linked Data datasets, SPARQL endpoints, or converting data from other legacy systems. A monolithic approach to constructing an aggregated dataset can be slow, fragile, and complex to implement. Dividing up tasks into smaller units of work can help parallelize the compilation of the data.
Solution

Create a number of separate processes that are each specialised to a particular task. Each process runs independently from each of the others, allowing each to be as simple or as complex as necessary. The processes all co-ordinate in the Progressive Enrichment of a single dataset, but are not triggered in any particular sequence. Instead each process looks for specific patterns in the underlying dataset, triggering the processing of the data, the results are then written back to the dataset.

Example(s)

An application is written to monitor twitter for uses of a particular hash tag. The application provides a simple RDF view of each tweet, including its author, text, timestamp. The RDF is written into a shared RDF dataset that serves as the Blackboard for a number of other processes. Additional processes are then written that query the dataset to discover new tweets. These processes carry out discrete tasks. The results of the tasks may trigger further processing by other tasks, or may be used to directly drive specific application behaviour. The tasks may include steps such as:

- Looking for profile information about the author of each tweet
- Attempting to find a FOAF profile for each author
- Extracting hash tags from new tweets
- Extracting URLs from individual tweets
- Discovering metadata associated with URLs mentioned in tweets

Discussion

The Blackboard pattern is an existing design pattern that has been used in a number of existing systems, it works well when applied to the task of aggregating RDF and Linked Data due to the ease with which a dataset can be enriched over time.

The RDF dataset used as the "blackboard" shared by the independent processes may be short-lived, e.g. an in-memory datastore used to respond to a particular request or task, or permanent, e.g. an on-going aggregation of data on a particular topic or theme.

The decomposition of the data aggregation & conversion tasks into smaller units makes it easier to explore different approaches for implementing the desired behaviour, e.g. to explore alternate technologies or algorithms. The overall result of the processes co-operating to compile and enrich the dataset can be extremely complex but without requiring any overall co-ordination effort. Additional processes steps, e.g. to acquire data from additional sources, can easily be added without impacting on the overall system, making the architecture extensible.

Related

- Progressive Enrichment
- Parallel Loading

Bounded Description

How can we generate a useful default description of a resource without having to enumerate all the properties or relations that are of interest?
Context

Application using semi-structured data sources should be tolerant of discovering unexpected data or missing properties of resources. For applications to be able to achieve this, there needs to be an approach to generating useful default descriptions of resources that don't require enumerating every property of interest. This behaviour is particularly useful for Linked Data browsers or similar applications that can have little or no expectation as to with which datasets they may be interacting.

Solution

Extract a sub-graph, or "bounded description", from a dataset that contains all of relevant properties and relationships associated with a resource.

Discussion

Bounded Descriptions take advantage of the graph structure of RDF in order to define simple graph operations that can be applied to any node in the graph. The operations yield a useful sub-set of the properties associated with the resource based on how they relate to the resource, rather than the specific RDF predicates that have been used.

There are a number of different types of bounded description that are in common use:

- **Datatype Property Description** -- retrieve all properties of a resource whose values are literals
- **Object Property Description** -- retrieve all properties of a resource whose values are resources, typically eliminating blank nodes
- **Concise Bounded Description** -- effectively the above two descriptions, but recursively include all properties of any blank nodes present in object properties
- **Symmetric Concise Bounded Description** -- as above but include statements where the resource being described is the object, rather than the subject

Many different variations of these basic descriptions are possible, especially when additional filtering is done to include, for example, properties that are useful for labelling (in a user interface).

In practice many common web application use cases can easily be fulfilled with one or more bounded description queries. The ability to use general purpose queries to build a user interface or otherwise drive application behaviour increases cacheability of the results: an application may end up using a small number of relatively general purpose queries that apply to a number of use cases.

Bounded descriptions can be implemented using SPARQL CONSTRUCT queries. SPARQL DESCRIBE queries are implemented using a Bounded Description that is built-in to the specific SPARQL processor being used. The most common approach is to use a Concise Bounded Description.

Further Reading

- Concise Bounded Description [http://www.w3.org/Submission/CBD/]

Composite Descriptions

How do we declare the underlying dataset for a page involving custom subsets or views of the data?
Context

When integrating data from heterogenous sources it is sometimes necessary to synthesise page URIs non-algorithmically from the underlying data. Alternatively views of data may be required that follow a clustering or structure that does not have a simple 1:1 correspondence with underlying data URIs.

Solution

Create data about your description pages and include foaf:topic and foaf:primaryTopic properties to link the page to the resources that it describes. When rendering these pages obtain the data describing the page then bring in descriptions of each resource referenced with foaf:topic and foaf:primaryTopic to build the base dataset for the page.

Example(s)

The BBC programme pages include information on a primary topic supplemented with additional data about other related topics. The data included on each page may vary depending on factors other than the type of resource being described.

Discussion

Most database driven pages on the Web involve more than one type of data and augment a base dataset with related information by using multiple database queries. In many cases it is possible for the publisher to anticipate these arrangements and describe them as linked data. This can reduce multiple queries to a single query describing the resource and any related resources. The resulting dataset can be passed directly to a templating system for rendering.

The topics associated with a page do not need to be closely related in the underlying data or even connected at all. The page description gathers together a group of resources according to the precise context specified by the publisher without reliance on particular relationships pre-existing in the data.

Changing the level of detail for classes of page or even of specified individual pages can be done simply by updating the description of those pages and allowing the templating system to work with the new dataset.

An additional benefit is that the page structure of the site can also be made queryable so it would be possible to discover which pages include information about a specific subject, thereby presenting the possibility of automatic cross-linking.

Follow Your Nose

*How do we find additional relevant data from the web?*

Context

When we retrieve some data from a URI, we are unlikely to have obtained all of the relevant data about that resource. Additional statements or useful extra context may be available from both the original source, as well as other third-party sources on the web.

Solution

Identify additional useful links within the available data, and then de-reference those URIs in turn to find the additional data.
Example(s)

The BBC Wildlife Finder application exposes data about biological species. By following links within the data we can find additional information about related species or its habitat from within the same dataset. By following links to dbpedia or other sources, we can find additional detail on the biology and distribution of the species.

Discussion

This pattern is at the core of the Linked Data approach. By giving resources URIs we make them part of the web, allowing a description of them to be retrieved by a simple HTTP GET request. By linking to other resources on the web, we allow applications to find more information by repeatedly following links to crawl the additional sources.

There are two main types of links that could be followed in an RDF graph:

- URIs of other resources -- See Also links to further documents, or URIs of other related resources. Follow these links to find more data.

- URIs of RDF terms -- links to definitions of properties, classes and other terms. Follow these links to find machine-readable descriptions of terms

An application is unlikely to want to blindly follow all links. For example applications will certainly want to place a limit on how many additional links it will want to fetch, e.g. one or two hops from the original resource. An application may also want to limit the data retrieved, e.g. by only following certain types of relationship or restricting the domains from which data will be retrieved. The former allows a more directed "crawl" to find related information, while the latter allows simple white/black-listing to only obtain data from trusted sources.

An application might also want to limit network traffic by performing Resource Caching. Parallel Retrieval can also improve performance

The retrieved data will often be parsed into one RDF graph that can then be queried or manipulated within the application. This "working set" might be cached as well as the original source descriptions, to allow for the fact that the same data may be repeatedly referenced.

Some additional processing may also be carried out on the retrieved data, e.g. to apply Smushing to combine all available data about a resource into a single description.

Related

- Missing Isn't Broken
- See Also
- Smushing
- Resource Caching
- Parallel Retrieval

Missing Isn't Broken

*How do we handle the potentially messy or incomplete data we use from the web?*
Context

In RDF anyone can say anything, anywhere. In other words anyone can make statements about a resource and publish that to the web for others to use. There is no requirement about how much data needs to be published: there are no validation rules that require a minimum amount of data. This means that data on the web may be of varying quality or of varying detail.

This variation is partly a factor of the flexibility of the model, but is really also a fact of life when dealing with any data or content found on the web: even within well-defined standards there may be varying levels of detail available.

How do we deal with this in our Linked Data applications?

Solution

Recognise that "missing isn't broken"

Example(s)

An application might chose to render as much of a FOAF profile (for example) as it can, even though individual profiles might be of varying details.

Discussion

This pattern is really just a restatement of Postel's Law: Be conservative in what you send; be liberal in what you accept. This advice is particularly applicable when dealing with any data or content obtained from the web. Applications ought to be tolerant of missing or invalid data and make best effort to process or render what is available.

In a Linked Data context this advice is particularly applicable as the flexibility of the RDF model means that there is greater chance for variation in detail across data sources. Rather than rely on schema or document validation, as in XML or relational database systems, to identify and reject data, applications should be designed to be more tolerant.

Of course an application may require some minimum data in order to do anything useful with some data. Although if a data publisher has followed the Label Everything pattern then at a minimum a data browser, for example, may still be able to render the name of the resource.

Unlike other approaches, where data is found to be missing, Linked Data provides additional opportunities for finding more data by supplementing the available data with additional sources, E.g. by using the Follow Your Nose pattern.

Related

- Follow Your Nose
- Label Everything

Further Reading

- Missing Isn't Broken
Named Query

How can details of the SPARQL protocol be hidden from clients?

Context

SPARQL protocol URLs quickly become complex when dealing with any non-trivial query. Very large queries can be so long that some clients or browsers may have issues with length of the URLs. The only solution in this case is to switch from a GET request to a POST request. But as a query is an idempotent operation it is better to use GET, with appropriate caching headers, rather than a POST.

In other circumstances a service might want to restrict the set of queries that can be invoked against a SPARQL endpoint. Or the use of SPARQL might be entirely hidden from calling clients. In both of those cases removal of direct access to the SPARQL endpoint may be desirable.

Solution

Assign a short URL to the SPARQL protocol request. The URL maps directly to a SPARQL query that is executed on request. Clients can use the short URL instead of the full SPARQL protocol request to extend the query.

Example(s)


One example query that clients might potentially execute is:

```sparql
SELECT ?uri ?homepage WHERE {
}
```

Rather than requiring clients to compose the full SPARQL protocol request for that URL it could instead be defined as a named query for the service. The query could be associated with the following URL: http://api.example.org/sparql/list-homepages. A GET request to that URL would be equivalent to the SPARQL protocol request to the endpoint, i.e. would execute the configured SPARQL query and return a response in one of the standard SPARQL protocol formats.

Discussion

Named queries is useful in a number of circumstances. Assigning short URLs to queries can remove issues with dealing with lengthy SPARQL queries that might get accidentally truncated in emails or be rejected by older HTTP clients or browsers. By providing tools for users of a service to create new named queries then a community can share and publish a useful set of queries.

Another benefit of binding queries to URLs, i.e. by creating new web resources, a service can implement additional optimisations that can improve response time of queries. E.g. query results might be generated asynchronously and the cached results supplied to clients rather than the query being executed on demand.

One way to protect a SPARQL endpoint is to reduce the legal set of queries to an approved list, e.g. that won't cause performance issues for the service. Named queries provide a way to provide a set of legal queries which are then bound to URLs. Direct access to the SPARQL endpoint can then be disabled, or limited to a white-listed set of client applications.
There are some additional nuances to consider when implementing this pattern. For example the SPARQL protocol could be extended to support Parameterised queries by injecting query string parameters into the query before it is executed. Additional parameters could be used to invoke additional kinds of pre- or post-processing behaviour including transformation of SPARQL protocol responses into alternate formats

Related
- Parameterised Query

Further Reading
- SPARQL Stored Procedure

Parallel Loading

*How can we reduce loading times for a web-accessible triple store?*

Context

It is quite common for triple stores to expose an HTTP based API to support data loading. E.g. via SPARQL 1.1 Update or the SPARQL 1.1. Uniform Protocol. It can be inefficient or difficult to POST very large datasets over HTTP, e.g. due to protocol time-outs, network errors, etc

Solution

Chunk the data to be loaded into smaller files and use a number of worker processes to submit data via parallel HTTP requests

Example(s)

Most good HTTP client libraries will support parallelisation of HTTP requests. E.g. PHP's curl_multi or Ruby's typhoeus library.

Discussion

Parallelization can improve any process. Because an RDF graph is a set of triples there is no ordering criteria for adding statements to a store. This means that it is usually possible to divide up an RDF data dump into a number of smaller files or chunks for loading via parallel POST requests.

This approach works best when the RDF data is made available as N-Triples, because the chunking can be done by simply splitting the file on line numbers. This isn't possible with RDF/XML or Turtle files that use prefixes or other syntax short-cuts.

The one caveat to this approach is if the data contains blank nodes. It is important that all statements about a single blank node are submitted in the same batch. Either avoid using bnodes, or split the file based on a Bounded Description of each resource.

Related
- Parallel Retrieval
Parallel Retrieval

*How can we improve performance of an application dynamically retrieving Linked Data?*

**Context**

An application that draws on data from the web may typically be retrieving a number of different resources. This is especially true if using the Follow Your Nose pattern to discover data.

**Solution**

Use several workers to make parallel GET requests, with each work writing into a shared RDF graph.

**Example(s)**

Most good HTTP client libraries will support parallelisation of HTTP requests. E.g. PHP's curl_multi or Ruby's typhoeus library.

**Discussion**

Parallelisation of HTTP requests can greatly reduce retrieval times, e.g. to time of the single longest GET request.

By combining this approach with Resource Caching of the individual responses, an application can maintain a local cache of the most requested data, which are then combined and parsed into a single RDF graph for driving application behaviour.

Parallelisation is particularly useful for AJAX based applications as browsers are particularly well optimized for making a large number of parallel HTTP requests.

**Related**

- Follow Your Nose
- Parallel Loading

Parameterised Query

*How to avoid continual regeneration and reparsing of SPARQL queries that differ only in a few bound variables?*

**Context**

Many applications continually execute a small number of queries which differ only in terms of a few parameters. For example generating a Bounded Description of a resource might involve a standard query that varies only by the resource URI being referenced. Other queries might vary based on date ranges, page offsets, etc. Re-generating queries as text strings can be fiddly and makes for messy application code. This is particularly true when the query then needs to be serialised and submitted over the SPARQL protocol: URL encoding issues can easily cause problems. In addition a query engine incurs extra overhead when repeatedly parsing the same query.
**Solution**

Define each repeated query as a query template which can be parsed once by the query engine. The query should define variables for those aspects of the query that might vary. The variables can then be bound to the query before execution. Supplying the values for these parameters will typically involve an engine specific API call.

**Example(s)**

Apache Jena uses a `QueryExecutionFactory` to support creation of queries. Query objects can be pre-compiled. An initial set of bindings for a query can be provided in order to create a specific `QueryExecution`. These initial bindings are simply a map from variable name to value.

**Discussion**

This pattern is well understood in the SQL database world: Prepared Statements have been in use for many years. SPARQL processors are now starting to add similar features, which simplifies working with queries from application code.

At present the SPARQL Protocol does not support specification of initial bindings as additional query string parameters. However some Linked Data publishing platforms have added support for parameterised queries, as extensions to the core protocol, allowing additional query parameters to be automatically injected into the query prior to execution. This makes it simpler for users to share queries and adjust their parameters.

SPARQL 1.1 provides a `BINDINGS` keyword which can be used to declare that certain variables should be injected into a graph pattern. This was added primarily to support federated query use cases, but can also be used to support some parameterisation of queries.

A map of named-value pairs which describe some initial query bindings can also be interpolated directly into a query by looking for appropriately named query variables. This interpolation has been done in several ways:

- Using some additional custom syntax to mark up the variable, e.g. `%{var}`. This means that the query is no longer valid SPARQL into variables have been substituted.

- By using the fact that SPARQL provides to naming syntaxes for variables, and defining `$var` to be those that are bound before execution and `?var` those that are bound during execution (or vice versa). This relies on local convention.

- By binding any query variable using any valid SPARQL syntax. The downside to this option is that additional external context is required to identify those variables that must be bound prior to execution. The other options allow these to be identified from just the query itself, although in practice it is often useful to be able to known the expect type, format or legal values for a parameter which will require additional configuration anyway.

Parameterised Queries are a core feature of the Named Query pattern.

**Related**

- Named Query

**Further Reading**

- SPARQL 1.1. Bindings
Resource Caching

How can an application that relies on loading data be more tolerant of network failures and/or reduce use of bandwidth

Context

Linked Data applications will typically need to discover and load data and schemas from the web. A user may request that extra data is displayed from specific locations, and the loading of a new data source may trigger loading of additional schemas, e.g. to discover labels for properties and types, or to pass to a reasoner for inferring additional data and relationships. Some resources and vocabularies may be very commonly used, e.g. the RDF, RDF Schema and OWL vocabularies, while others may only be encountered during run-time.

Solution

Build a local cache of retrieved resources, refreshing the cache only when source data has changed.

Discussion

Retrieving resources from the web, like any other network access, is prone to failure. Repeated fetching of the same resources is waste-ful of bandwidth on the client and the server: a large number of clients can easily overload resources on a system serving up popular vocabularies.

Applications should cache remote resources wherever possible. The cache may be handled entirely in-memory but with sufficient permissions and access to the local file-system an application could also build a persistent cache. Desktop application may ship with a pre-seeded cache of commonly retrieved resources such as ontologies. Efficient use of HTTP request can ensure that cached versions need only be updated when the remote copy of the resource has been updated. Certain vocabularies, e.g. RDF Schema, will only change rarely, if at all. These could be cached for longer periods, if not permanently.

Ideally applications should provide configuration to support the user in managing the amount of local resources (memory or disk space) that can be used by the cache. Control over the location in which cached data will be stored is also useful.

Related

• Follow Your Nose
• Parallel Retrieval

Schema Annotation

How can application-specific processing rules be externalized?

Context

Data driven applications typically end up with built-in processing rules for handling particular types of data, e.g. validation constraints, preferences for specific properties or types, etc. These rules are often encapsulated in procedural code, making them difficult to change. Externalizing these rules as declarative configuration can make an application easier to customize. How can this be achieved with applications that consume Linked Data?
Solution

Externalize constraints using annotation properties that are used to drive processing rules or constraints by annotating classes and properties in a vocabulary

Example(s)

```turtle
ex:RequiredProperty a rdfs:Property;
    rdfs:comment "must be completed on data entry form".

ex:IgnoredProperty a rdfs:Property;
    rdfs:comment "never shown when displaying data".

<http://xmlns.com/foaf/0.1/name>
    a ex:RequiredProperty.

<http://xmlns.com/foaf/0.1/dnaChecksum>
    a ex:IgnoredProperty.
```

Discussion

Simple annotations of classes and properties is a simple and easy way to externalize some common types of application configuration. RDF vocabularies are easily extended with additional properties, making them suitable for extension in this way. Using this approach applications can be very easily tailored to work with a range of different vocabularies.

Annotations may encode a wide range of configuration options including: display preferences, validation constraints, identifier assignment rules for classes, and local labelling for classes and properties. Annotation may even be used to tailor inferencing over specific vocabularies to allow for more local customisation and control over how inferencing is applied; for example a local schema annotation might declare that two classes were equivalent, or that a specific property is an inverse-functional-property, triggering data to be merged.

Schema annotations would typically form part of the overall application configuration and would be applied locally, rather than being published to the open web.

Related

• Annotation

Smushing

*How do we merge data about resources that may not be consistently identified?*

Context

It will often be the case that different data publishers have used different identifiers for the same resource. In some cases there may be direct Equivalence Links between resources. In others their equivalence might be inferred based on other data, e.g. common properties.
How can we merge statements made about these distinct resources into a single description?

Solution

Apply the technique of "smushing" to manipulate an RDF graph containing the descriptions of each of the resources. Broadly a smushing algorithm will consist of the following steps:

- Decide on the URI for the resource that will hold the final description, i.e. the target resource -- this could be one randomly selected from available URIs, or one from a local dataset
- Identify all equivalent resources -- i.e. by finding Equivalence Links such as owl:sameAs statements, or by property values that indicate that two resources are similar (e.g. Inverse Functional Properties, see below).
- Iterate over the equivalent resources and for each RDF statement for which it is the subject, assert a new statement with the same predicate and object but using the target resource as the subject
- Iterate over the equivalent resources and for each RDF statement for which it is the object, assert a new statement with the same subject and predicate but using the target resource as the object

The end result will be an modified RDF graph with all properties of the equivalent resources being "copied" to the target resource. In addition, any references to the equivalent resources will also be made to the target resource.

By applying this to all resources in a graph, the available data can be normalized into a consistent set of descriptions based on a known set of resources. An application may then generate a Bounded Description of any resource and guarantee that it will include all available data.

Example(s)

Assume we start with the following graph, which contains two equivalent resources, as defined by an owl:sameAs link.

```
<http://example.com/product/6>
  rdfs:label "Camera";
  owl:sameAs <http://example.org/cameras/10>.

<http://example.org/cameras/10>
  ex:manufacturer <http://example.org/company/5>.

<http://example.org/company/5>.
```

Assuming we want to collate all data around resources from example.com, we can apply smushing to create the following graph:

```
<http://example.com/product/6>
  rdfs:label "Camera";
```
owl:sameAs <http://example.org/cameras/10>;
ex:manufacturer <http://example.org/company/5>.

<http://example.org/cameras/10>
ex:manufacturer <http://example.org/company/5>.

<http://example.org/company/5>.

We can also tidy up the graph to remove statements about the equivalent resources, leaving:

<http://example.com/product/6>
  rdfs:label "Camera";
  owl:sameAs <http://example.org/cameras/10>;
ex:manufacturer <http://example.org/company/5>.

<http://example.org/company/5>.

Discussion

Smushing is essentially a process of inference: by using available data we create new statements. Any OWL reasoner will carry out this kind of data merging automatically based on the available data and schema/ontology without the need for custom code. Applications that are using a triple store that applies inferencing by default will not need to use this approach. However for applications that don't need a full inferencing engine, or need only lightweight merging of data, then a custom smushing algorithm can achieve the same goal.

There are several different variations on algorithm described above. For example, applications might vary in how they nominate the target resource. Typically though this will be based on a preferred URI. Algorithms can also be divided into those that preserve the original statements, e.g. so that the equivalent resources remain in the source RDF graph, or whether their statements are removed from the graph to leave only a normalized description. Applications could also use Named Graphs to separately stored the "smushed" view of the data, preserving the original data in another graph or triple store.

As noted above there are also several ways to identify equivalent resources. Equivalence Links are an obvious approach. Other cues can also be used including the use of Inverse Functional Properties. An inverse functional property is simply a property whose value uniquely identifies a resource, such as Literal Keys.

An application is also free to apply it's own rules about what constitutes "equivalence". For example an application may decide to merge together resources with similar property values, even if those properties are not declared as Inverse Functional Properties. This allows for local customization of smushing rules, but runs the risk of generating false positives. One way to apply these custom rules is to use local Schema Annotations to declare specific properties as being equivalent. This has the benefit of working with both custom code and OWL reasoners.

Smushing is often used to normalize an RDF graph resulting from a Follow Your Nose approach to data discovery.
Related

• Equivalence Links
• Follow Your Nose

Further Reading

• Smushing
• RDF Smushing
• Smushing Algorithms

Transformation Query

*How can we normalize or transform some RDF data so that it conforms to a preferred model?*

Context

There are a broad range of different vocabularies in use on the Linked Data web. While there has been convergence on common vocabularies in a number of domains, there will always be some variations in how data is presented, e.g. using slightly different modelling styles and vocabularies. Schemas will also evolve over time and certain properties may be refined or replaced. How can a consuming application normalize these different models into a single preferred representation that matches expectations or requirements of the application code?

Solution

Use a SPARQL CONSTRUCT query, or collection of queries, to generate a normalized view of the data, saving the results back into the triple store.

Example(s)

The following query normalizes any one of three different naming or labelling properties into rdfs:label properties.

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
CONSTRUCT {
  ?s rdfs:label ?o.
}
WHERE {
  { ?s foaf:name ?o. }
  UNION
  { ?s vCard:FN ?o. }
  UNION
  { ?s vCard:FN ?o. }
  UNION
  { ?s rdfs:label ?o. }
  UNION
  { ?s rdfs:label ?o. }
  UNION
  { ?s rdfs:label ?o. }
}
```
Writing application code to normalize data patterns can be tedious and difficult to maintain in the face of changes. SPARQL CONSTRUCT queries provide a way to generate arbitrary RDF graphs from existing data. They therefore provide a way to carry out transformations on RDF graph. These transformations can be used to infer new data based on existing patterns and this covers the generation of normalized data models. SPARQL therefore provides a declarative syntax for describing graph transformations. The increased expressivity of SPARQL 1.1 will allow more complex transformations to be specified.

An application may apply one or more Transformation Queries to its data source either during execution, e.g. to extract a graph of a known structure, or during assembly of the underlying dataset, e.g. as part of the Blackboard pattern.

Each transformation query may cover one specific normalization task. However, as shown in the example above, several operations can be combined using a UNION query. This allows the graph pattern of the query to match for a number of different variants, resulting in the generation of a single standard output graph.

**URI Resolver**

*How can we customize the application behaviour associated with resolving (de-referencing) a URI into RDF statements?*

**Context**

Linked Data applications typically acquire additional relevant data by adopting Follow Your Nose behaviour: any URI in a graph may be assumed to be de-referencable to obtain additional data.

However in practice simple de-referencing, i.e. performing a GET request on a URI, is not always desirable. For example:

- An mobile application may need to work in an off-line mode where remote data is not available
- Continuous testing scenarios may need to rely on predictable data for driving test assertions and, in addition, may need to be executable in a self-contained environment without use of network services
- Security constraints may require network requests to be routed via an intermediary service
- A local mirror may be available which can provide a better quality of service
- A remote service may be intermittently or completely unavailable, requiring a local mirror to be substituted on either a temporary or permanent basis

**Solution**

Application code should address de-referencing requests to URI resolver. Broadly, a URI resolver is a function that maps from a URI to a stream from which RDF triples can be consumed. A URI resolver might consist of an application component or could be deployed as a network addressable service (i.e. a proxy server).
Application code should defer to the URI resolver in order to source RDF statements and provide configuration options to specify which URI resolver (e.g. implementation or service location) should be used. Simple de-referencing behaviour may still be used as fallback behaviour if no URI resolver is available.

**Example(s)**

A Linked Data browser loads and displays resources as directed by user behaviour, e.g. clicking on links in the user interface. A user selects to view a resource. When a user requests that the browser displays a resource, `http://example.org/person/1`, instead of performing a GET request on the resource the browser invokes a pre-configured URI resolver to retrieve the description of the resource.

The URI resolver has been set up to direct requests matching a pattern of `http://example.org/*` to a local triple store that contains a mirror of the remote data. However when the user visits `http://other.example.org/document/123` the URI resolver does not have any prior knowledge of the resource and falls back to a simple GET request on the resource URI.

In neither case does the browser (or the user) need to know how the description was actually retrieved.

**Discussion**

Adding some extra indirection around the de-referencing of URIs into RDF statements provides some much needed flexibility when dealing with network issues such as intermittently available connections; unreliable remote services; and security constraints. Applications that support the configuration of URI resolvers provide options for customising and optimising application behaviour based on local requirements.

URI resolvers are not a new concept and have been used in many different systems. SGML and XML processing pipelines typically support URI resolver components to allow resources to be cached locally or even bundled with an application. More broadly, HTTP proxy servers fulfill the role for general web requests.

The indirection offered by URI resolvers make them an ideal location in which to provide additional behaviour. For example all of the following can be implemented using a URI resolver component:

- Caching of RDF descriptions as they are retrieved, e.g. in an in-memory, file system, or document store
- Substitution of a local mirror of the data in preference for the remote version
- Substitution of a local mirror of the data in preference for the remote version, but only where the remote service is unavailable
- Serving of a fixed response, regardless of URI (e.g. to support testing scenarios)
- Retrieval of both the remote description of a resource plus local annotations to mix public and private data
- Parallel retrieval of the description of a resource that is spread across any combination of local or remote locations
- Provision of reasoning over retrieved data to augment data against a vocabulary
- Provision of support for resolution of non-HTTP URI schemes
- On-demand conversion of non-RDF data into RDF statements
With suitable configuration, URI resolvers can potentially be chained together to create a de-referencing pipeline that can deliver some complex application behaviours with a simple framework.

There are some Linked Data applications that provide URI resolver services, this includes generic Linked Data browsers. At their simplest the browsers simply provide additional HTML presentation of retrieved data. But in some cases the retrieved data is both directly accessible (i.e. the service acts as a proxy) and may be supplemented with local caches, annotation, or inferencing, as outlined above. To support de-referencing typically use a Rebased URI

**Related**

- Follow Your Nose

**Further Reading**

- Diverted URI pattern
- The Jena FileManager and LocationMapper
- Entity management in XML applications
JSON-LD

<table>
<thead>
<tr>
<th>Filename extension</th>
<th>.jsonld</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet media type</td>
<td>application/ld+json</td>
</tr>
<tr>
<td>Type of format</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>Container for</td>
<td>Linked Data</td>
</tr>
<tr>
<td>Extended from</td>
<td>JSON</td>
</tr>
<tr>
<td>Standard</td>
<td>JSON-LD 1.0 (<a href="http://www.w3.org/TR/json-ld/">http://www.w3.org/TR/json-ld/</a>) / JSON-LD 1.0 API (<a href="http://www.w3.org/TR/json-ld-api/">http://www.w3.org/TR/json-ld-api/</a>)</td>
</tr>
<tr>
<td>Open format?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

JSON-LD 1.0 - A JSON-based Serialization for Linked Data

<table>
<thead>
<tr>
<th>Status</th>
<th>W3C Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year started</td>
<td>2010</td>
</tr>
<tr>
<td>Editors</td>
<td>Manu Sporny, Gregg Kellogg, Markus Lanthaler</td>
</tr>
<tr>
<td>Authors</td>
<td>Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Niklas Lindström</td>
</tr>
<tr>
<td>Base standards</td>
<td>JSON, RDF</td>
</tr>
<tr>
<td>Domain</td>
<td>Semantic Web, Data Serialization</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>JSON-LD</td>
</tr>
<tr>
<td>Website</td>
<td>JSON-LD 1.0 (<a href="http://www.w3.org/TR/json-ld/">http://www.w3.org/TR/json-ld/</a>)</td>
</tr>
</tbody>
</table>

JSON-LD 1.0 Processing
JSON-LD (JavaScript Object Notation for Linked Data), is a method of encoding Linked Data using JSON. It was a goal to require as little effort as possible from developers to transform their existing JSON to JSON-LD.\[1\] This allows data to be serialized in a way that is similar to traditional JSON.\[2\] It is a World Wide Web Consortium Recommendation. It was initially developed by the JSON for Linking Data Community Group (http://json-ld.org/) before being transferred to the RDF Working Group\[3\] for review, improvement, and standardization.\[4\]

## Contents

- Design
- Example
- Use
- See also
- References
- External links

### Design

JSON-LD is designed around the concept of a "context" to provide additional mappings from JSON to an RDF model. The context links object properties in a JSON document to concepts in an ontology. In order to map the JSON-LD syntax to RDF, JSON-LD allows values to be coerced to a specified type or to be tagged with a language. A context can be embedded directly in a JSON-LD document or put into a separate file and referenced from different documents (from traditional JSON documents via an HTTP Link header).
The example above describes a person, based on the FOAF vocabulary. First, the two JSON properties name and homepage and the type Person are mapped to concepts in the FOAF vocabulary and the value of the homepage property is specified to be of the type @id, i.e., it is specified to be an IRI in the context definition. Based on the RDF model, this allows the person described in the document to be unambiguously identified by an IRI. The use of resolvable IRIs allows RDF documents containing more information to be transcluded which enables clients to discover new data by simply following those links; this principle is known as Follow Your Nose.[5]

By having all data semantically annotated as in the example, an RDF processor can identify that the document contains information about a person (@type) and if the processor understands the FOAF vocabulary it can determine which properties specify the person’s name and homepage.

Use

The encoding is used by Schema.org[6], Google Knowledge Graph[7], and used mostly for search engine optimization activities. It has also been used for applications such as biomedical informatics,[8] and representing provenance information.[9] It is also the basis of ActivityStreams, a format for "the exchange of information about potential and completed activities"[10], and is used in ActivityPub, the federated social networking protocol[11].

See also

- Hypertext Application Language

References

3. RDF Working Group (http://www.w3.org/2011/rdf-wg/wiki/Main_Page) This Working Group ended its activities on 1 July 2014 and is now closed.


External links

- JSON-LD.org (http://json-ld.org/)


This page was last edited on 20 October 2018, at 04:31 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.
The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. It is also used in knowledge management applications.

RDF was adopted as a W3C recommendation in 1999. The RDF 1.0 specification was published in 2004, the RDF 1.1 specification in 2014.

## Contents

- Overview
- History
- RDF topics
  - Vocabulary
    - Classes
      - rdf
      - rdfs
    - Properties
      - rdf
      - rdfs
  - Serialization formats
  - Resource identification
  - Statement reification and context
  - Query and inference languages
  - Validation and description
- Examples
  - Example 1: Description of a person named Eric Miller
  - Example 2: The postal abbreviation for New York
  - Example 3: A Wikipedia article about Tony Benn
- Applications
- See also
- Notes
- References
- Further reading
- External links

## Overview
The RDF data model[2] is similar to classical conceptual modeling approaches (such as entity–relationship or class diagrams). It is based on the idea of making statements about resources (in particular web resources) in expressions of the form subject–predicate–object, known as triples. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, and expresses a relationship between the subject and the object.

For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue". Therefore, RDF uses subject instead of object (or entity) in contrast to the typical approach of an entity–attribute–value model in object-oriented design: entity (sky), attribute (color), and value (blue).

RDF is an abstract model with several serialization formats (i.e. file formats), so the particular encoding for resources or triples varies from format to format.

This mechanism for describing resources is a major component in the W3C's Semantic Web activity: an evolutionary stage of the World Wide Web in which automated software can store, exchange, and use machine-readable information distributed throughout the Web, in turn enabling users to deal with the information with greater efficiency and certainty. RDF's simple data model and ability to model disparate, abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity.

A collection of RDF statements intrinsically represents a labeled, directed multi-graph. This in theory makes an RDF data model better suited to certain kinds of knowledge representation than are other relational or ontological models. However, in practice, RDF data is often stored in relational database or native representations (also called Triplestores—or Quad stores, if context such as the named graph is also stored for each RDF triple).[3]

As RDFS and OWL demonstrate, one can build additional ontology languages upon RDF.

History

The initial RDF design, intended to "build a vendor-neutral and operating system-independent system of metadata,"[4] derived from the W3C's Platform for Internet Content Selection (PICS), an early web content labelling system,[5] but the project was also shaped by ideas from Dublin Core, and from the Meta Content Framework (MCF),[4] which had been developed during 1995–1997 by Ramanathan V. Guha at Apple and Tim Bray at Netscape.[6]

A first public draft of RDF appeared in October 1997,[7][8] issued by a W3C working group that included representatives from IBM, Microsoft, Netscape, Nokia, Reuters, SoftQuad, and the University of Michigan.[5]

In 1999, the W3C published the first recommended RDF specification, the Model and Syntax Specification ("RDF M&S").[9] This described RDF's data model and an XML serialization.[10]

Two persistent misunderstandings about RDF developed at this time: firstly, due to the MCF influence and the RDF "Resource Description" initialism, the idea that RDF was specifically for use in representing metadata; secondly that RDF was an XML format rather than a data model, and only the RDF/XML serialisation being XML-based. RDF saw little take-up in this period, but there was significant work done in Bristol, around ILRT at Bristol University and HP Labs, and in Boston at MIT. RSS 1.0 and FOAF became exemplar applications for RDF in this period.

This series was superseded in 2014 by the following six "RDF 1.1" documents: "RDF 1.1 Primer," "RDF 1.1 Concepts and Abstract Syntax," "RDF 1.1 XML Syntax," "RDF 1.1 Semantics," "RDF Schema 1.1," and "RDF 1.1 Test Cases."

RDF topics

Vocabulary

The vocabulary defined by the RDF specification is as follows:

Classes

rdf
- rdf:XMLLiteral – the class of XML literal values
- rdf:Property – the class of properties
- rdf:Statement – the class of RDF statements
- rdf:Alt, rdf:Bag, rdf:Seq – containers of alternatives, unordered containers, and ordered containers (rdfs:Container is a super-class of the three)
- rdf:List – the class of RDF Lists
- rdf:nil – an instance of rdf:List representing the empty list

rdfs
- rdfs:Resource – the class resource, everything
- rdfs:Literal – the class of literal values, e.g. strings and integers
- rdfs:Class – the class of classes
- rdfs:Datatype – the class of RDF datatypes
- rdfs:Container – the class of RDF containers
- rdfs:ContainerMembershipProperty – the class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of rdfs:member

Properties

rdf
- rdf:type – an instance of rdf:Property used to state that a resource is an instance of a class
- rdf:first – the first item in the subject RDF list
- rdf:rest – the rest of the subject RDF list after rdf:first
- rdf:value – idiomatic property used for structured values
- rdf:subject – the subject of the subject RDF statement
- rdf:predicate – the predicate of the subject RDF statement
- rdf:object – the object of the subject RDF statement

rdf:Statement, rdf:subject, rdf:predicate, rdf:object are used for reification (see below).

rdfs
- rdfs:subClassOf – the subject is a subclass of a class
 Serialization formats
 Several common serialization formats are in use, including:

- **Turtle**[27] a compact, human-friendly format.
- **N-Triples,[28]** a very simple, easy-to-parse, line-based format that is not as compact as Turtle.
- **N-Quads,[29][30]** a superset of N-Triples, for serializing multiple RDF graphs.
- **JSON-LD,[31]** a JSON-based serialization.
- **N3 or Notation3,** a non-standard serialization that is very similar to Turtle, but has some additional features, such as the ability to define inference rules.
- **RDF/XML,[32]** an XML-based syntax that was the first standard format for serializing RDF.
- **RDF/JSON,[33]** an alternative syntax for expressing RDF triples using a simple JSON notation.

RDF/XML is sometimes misleadingly called simply RDF because it was introduced among the other W3C specifications defining RDF and it was historically the first W3C standard RDF serialization format. However, it is important to distinguish the RDF/XML format from the abstract RDF model itself. Although the RDF/XML format is still in use, other RDF serializations are now preferred by many RDF users, both because they are more human-friendly,[34] and because some RDF graphs are not representable in RDF/XML due to restrictions on the syntax of XML QNames.

With a little effort, virtually any arbitrary XML may also be interpreted as RDF using GRDDL (pronounced 'griddle'), Gleaning Resource Descriptions from Dialects of Languages.

RDF triples may be stored in a type of database called a triplestore.

Resource identification
The subject of an RDF statement is either a uniform resource identifier (URI) or a blank node, both of which denote resources. Resources indicated by blank nodes are called anonymous resources. They are not

- `rdfs:subPropertyOf` – the subject is a subproperty of a property
- `rdfs:domain` – a domain of the subject property
- `rdfs:range` – a range of the subject property
- `rdfs:label` – a human-readable name for the subject
- `rdfs:comment` – a description of the subject resource
- `rdfs:member` – a member of the subject resource
- `rdfs:seeAlso` – further information about the subject resource
- `rdfs:isDefinedBy` – the definition of the subject resource

This vocabulary is used as a foundation for RDF Schema where it is extended.
directly identifiable from the RDF statement. The predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a Unicode string literal. As of RDF 1.1 resources are identified by IRI's. IRI is a generalization of URI.[35]

In Semantic Web applications, and in relatively popular applications of RDF like RSS and FOAF (Friend of a Friend), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources. In fact, the URI that names a resource does not have to be dereferenceable at all. For example, a URI that begins with "http:" and is used as the subject of an RDF statement does not necessarily have to represent a resource that is accessible via HTTP, nor does it need to represent a tangible, network-accessible resource — such a URI could represent absolutely anything. However, there is broad agreement that a bare URI (without a # symbol) which returns a 300-level coded response when used in an HTTP GET request should be treated as denoting the internet resource that it succeeds in accessing.

Therefore, producers and consumers of RDF statements must agree on the semantics of resource identifiers. Such agreement is not inherent to RDF itself, although there are some controlled vocabularies in common use, such as Dublin Core Metadata, which is partially mapped to a URI space for use in RDF. The intent of publishing RDF-based ontologies on the Web is often to establish, or circumscribe, the intended meanings of the resource identifiers used to express data in RDF. For example, the URI:

```
http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Merlot
```

is intended by its owners to refer to the class of all Merlot red wines by vintner (i.e., instances of the above URI each represent the class of all wine produced by a single vintner), a definition which is expressed by the OWL ontology — itself an RDF document — in which it occurs. Without careful analysis of the definition, one might erroneously conclude that an instance of the above URI was something physical, instead of a type of wine.

Note that this is not a 'bare' resource identifier, but is rather a URI reference, containing the '#' character and ending with a fragment identifier.

**Statement reification and context**

The body of knowledge modeled by a collection of statements may be subjected to reification, in which each statement (that is each triple subject-predicate-object altogether) is assigned a URI and treated as a resource about which additional statements can be made, as in "Jane says that John is the author of document X". Reification is sometimes important in order to deduce a level of confidence or degree of usefulness for each statement.

In a reified RDF database, each original statement, being a resource, itself, most likely has at least three additional statements made about it: one to assert that its subject is some resource, one to assert that its predicate is some resource, and one to assert that its object is some resource or literal. More statements about the original statement may also exist, depending on the application’s needs.

Borrowing from concepts available in logic (and as illustrated in graphical notations such as conceptual graphs and topic maps), some RDF model implementations acknowledge that it is sometimes useful to group statements according to different criteria, called situations, contexts, or scopes, as discussed in articles by RDF specification co-editor Graham Klyne.[36][37] For example, a statement can be associated with a context, named by a URI, in order to assert an "is true in" relationship. As another example, it is sometimes convenient to group statements by their source, which can be identified by a URI, such as the URI of a particular RDF/XML document. Then, when updates are made to the source, corresponding statements can be changed in the model, as well.
Implementation of scopes does not necessarily require fully reified statements. Some implementations allow a single scope identifier to be associated with a statement that has not been assigned a URI, itself.\[38\][39] Likewise named graphs in which a set of triples is named by a URI can represent context without the need to reify the triples.\[40\]

Query and inference languages

The predominant query language for RDF graphs is SPARQL. SPARQL is an SQL-like language, and a recommendation of the W3C as of January 15, 2008.

An example of a SPARQL query to show country capitals in Africa, using a fictional ontology.

```sparql
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital ?country
WHERE {
  ?x ex:cityname ?capital ;
  ex:isCapitalOf ?y .
  ?y ex:countryname ?country ;
  ex:isInContinent ex:Africa .
}
```

Other non-standard ways to query RDF graphs include:

- RDQL, precursor to SPARQL, SQL-like
- Versa, compact syntax (non–SQL-like), solely implemented in 4Suite (Python).
- RQL, one of the first declarative languages for uniformly querying RDF schemas and resource descriptions, implemented in RDFSuite.\[41\]
- SeRQL, part of Sesame
- XUL has a template element in which to declare rules for matching data in RDF. XUL uses RDF extensively for databinding.

Validation and description

There are several proposals to validate and describe RDF:

- SHACL (Shapes Constraint Language) \[42\] is expresses constraints on RDF Graphs. SHACL is divided in two parts: SHACL Core and SHACL-SPARQL. SHACL Core consists of a list of built-in constraints such as cardinality, range of values and many others. SHACL-SPARQL consists of all features of SHACL Core plus the advanced features of SPARQL-based constraints and an extension mechanism to declare new constraint components.
- ShEx (Shape Expressions) \[43\] is a concise language for RDF validation and description.

Examples

Example 1: Description of a person named Eric Miller\[44\]

The following example is taken from the W3C website\[44\] describing a resource with statements "there is a Person identified by http://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is e.miller123(at)example (changed for security purposes), and whose title is Dr."

The resource "http://www.w3.org/People/EM/contact#me" is the subject.

The objects are:

- "Eric Miller" (with a predicate "whose name is"),
- mailto:e.miller123(at)example (with a predicate "whose email address is"), and
- "Dr." (with a predicate "whose title is").

The subject is a URI.

The predicates also have URIs. For example, the URI for each predicate:

- "whose name is" is http://www.w3.org/2000/10/swap/pim/contact#fullName,
- "whose email address is" is http://www.w3.org/2000/10/swap/pim/contact#mailbox,
- "whose title is" is http://www.w3.org/2000/10/swap/pim/contact#personalTitle.

In addition, the subject has a type (with URI http://www.w3.org/1999/02/22-rdf-syntax-ns#type), which is person (with URI http://www.w3.org/2000/10/swap/pim/contact#Person).

Therefore, the following "subject, predicate, object" RDF triples can be expressed:

- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#fullName, "Eric Miller"
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#mailbox, mailto:e.miller123(at)example
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/2000/10/swap/pim/contact#personalTitle, "Dr."
- http://www.w3.org/People/EM/contact#me, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, http://www.w3.org/2000/10/swap/pim/contact#Person

In standard N-Triples format, this RDF can be written as:

```
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/contact#fullName> "Eric Miller" .
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/contact#mailbox> mailto:e.miller123(at)example .
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/contact#personalTitle> "Dr." .
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> http://www.w3.org/2000/10/swap/pim/contact#Person .
```

Equivalently, it can be written in standard Turtle (syntax) format as:

```
@prefix eric: <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
eric:me contact:mailbox <mailto:e.miller123(at)example> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

Or, it can be written in RDF/XML format as:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#" xmlns:eric="http://www.w3.org/People/EM/contact#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:mailbox rdf:resource="mailto:e.miller123(at)example"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:personalTitle>Dr.</contact:personalTitle>
  </rdf:Description>
</rdf:RDF>
```
Example 2: The postal abbreviation for New York

Certain concepts in RDF are taken from logic and linguistics, where subject-predicate and subject-predicate-object structures have meanings similar to, yet distinct from, the uses of those terms in RDF. This example demonstrates:

In the English language statement 'New York has the postal abbreviation NY', 'New York' would be the subject, 'has the postal abbreviation' the predicate and 'NY' the object.

Encoded as an RDF triple, the subject and predicate would have to be resources named by URIs. The object could be a resource or literal element. For example, in the N-Triples form of RDF, the statement might look like:

```
```

In this example, "urn:x-states:New%20York" is the URI for a resource that denotes the US state New York, "http://purl.org/dc/terms/alternative" is the URI for a predicate (whose human-readable definition can be found at here [45]), and "NY" is a literal string. Note that the URIs chosen here are not standard, and don't need to be, as long as their meaning is known to whatever is reading them.

Example 3: A Wikipedia article about Tony Benn

In a like manner, given that "http://en.wikipedia.org/wiki/Tony_Benn" identifies a particular resource (regardless of whether that URI could be traversed as a hyperlink, or whether the resource is actually the Wikipedia article about Tony Benn), to say that the title of this resource is "Tony Benn" and its publisher is "Wikipedia" would be two assertions that could be expressed as valid RDF statements. In the N-Triples form of RDF, these statements might look like the following:

```
```

To an English-speaking person, the same information could be represented simply as:

```
The title of this resource, which is published by Wikipedia, is 'Tony Benn'
```

However, RDF puts the information in a formal way that a machine can understand. The purpose of RDF is to provide an encoding and interpretation mechanism so that resources can be described in a way that particular software can understand it; in other words, so that software can access and use information that it otherwise couldn’t use.

Both versions of the statements above are wordy because one requirement for an RDF resource (as a subject or a predicate) is that it be unique. The subject resource must be unique in an attempt to pinpoint the exact resource being described. The predicate needs to be unique in order to reduce the chance that the idea of Title or Publisher will be ambiguous to software working with the description. If the software recognizes http://purl.org/dc/elements/1.1/title (a specific definition for the concept of a title established by the Dublin Core Metadata Initiative), it will also know that this title is different from a land title or an honorary title or just the letters t-i-t-l-e put together.
The following example, written in Turtle, shows how such simple claims can be elaborated on, by combining multiple RDF vocabularies. Here, we note that the primary topic of the Wikipedia page is a "Person" whose name is "Tony Benn":

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://en.wikipedia.org/wiki/Tony_Benn>
   dc:publisher "Wikipedia" ;
   dc:title "Tony Benn" ;
   foaf:primaryTopic [ 
      a foaf:Person ;
      foaf:name "Tony Benn" ] .
```

Applications

- **DBpedia** – Extracts facts from Wikipedia articles and publishes them as RDF data.
- **YAGO** – Similar to DBpedia extracts facts from Wikipedia articles and publishes them as RDF data.
- **Wikidata** – Collaboratively edited knowledge base hosted by the Wikimedia Foundation.
- **Creative Commons** – Uses RDF to embed license information in web pages and mp3 files.
- **FOAF (Friend of a Friend)** – designed to describe people, their interests and interconnections.
- **Haystack client** – Semantic web browser from MIT CS & AI lab.[46]
- **IDEAS Group** – developing a formal 4D ontology for Enterprise Architecture using RDF as the encoding.[47]
- **Microsoft shipped a product, Connected Services Framework,[48]** which provides RDF-based Profile Management capabilities.
- **MusicBrainz** – Publishes information about Music Albums.[49]
- **NEPOMUK**, an open-source software specification for a Social Semantic desktop uses RDF as a storage format for collected metadata. NEPOMUK is mostly known because of its integration into the KDE SC 4 desktop environment.
- **Press Association** is a news agency in the UK. They use ontologies to dynamically identify and link their NoSQL data to do semantic publishing but in a dynamic, rules based way that creates custom content on the fly.[50]
- **RDF Site Summary** – one of several "RSS" languages for publishing information about updates made to a web page; it is often used for disseminating news article summaries and sharing weblog content.
- **Simple Knowledge Organization System (SKOS)** – a KR representation intended to support vocabulary/thesaurus applications
- **SIOC (Semantically-Interlinked Online Communities)** – designed to describe online communities and to create connections between Internet-based discussions from message boards, weblogs and mailing lists.[51]
- **Smart-M3** – provides an infrastructure for using RDF and specifically uses the ontology agnostic nature of RDF to enable heterogeneous mashing-up of information.[52]

Some uses of RDF include research into social networking. It will also help people in business fields understand better their relationships with members of industries that could be of use for product placement.[53] It will also help scientists understand how people are connected to one another.

RDF is being used to have a better understanding of road traffic patterns. This is because the information regarding traffic patterns is on different websites, and RDF is used to integrate information from different sources on the web. Before, the common methodology was using keyword searching, but this method is problematic because it does not consider synonyms. This is why ontologies are useful in this situation. But one of the issues that comes up when trying to efficiently study traffic is that to fully understand traffic, concepts related to people, streets, and roads must be well understood. Since these are human concepts, they require the addition of fuzzy logic. This is because values that are useful when describing roads, like slipperiness, are not precise concepts and cannot be measured. This would imply that the best solution would incorporate both fuzzy logic and ontology.[54]
See also

Notations for RDF

- TRiG
- TRIX
- RDF/XML
- RDFa
- JSON-LD

Similar concepts

- Entity–attribute–value model
- Graph theory – an RDF model is a labeled, directed multi-graph.
- Website Parse Template
- Tagging
- SciCrunch
- Semantic network

Other (unsorted)

- Associative model of data
- Business Intelligence 2.0 (BI 2.0)
- Data portability
- EU Open Data Portal
- Folksonomy
- Life Science Identifiers
- Swoogle
- Universal Networking Language (UNL)
- VoID

Notes


31. "JSON-LD 1.0: A JSON-based Serialization for Linked Data" (http://www.w3.org/TR/json-ld/). W3C.


35. RDF 1.1 Concepts and Abstract Syntax https://www.w3.org/TR/rdf11-concepts/

References


Further reading

- W3C's RDF at W3C (http://www.w3.org/RDF/): specifications, guides, and resources
- RDF Semantics (http://www.w3.org/TR/2004/REC-rdf-mt-20040210/): specification of semantics, and complete systems of inference rules for both RDF and RDFS

External links

- Resource Description Framework (https://curlie.org/Reference/Libraries/Library_and_Information_Science/Technical_Services/Cataloguing/Metadata/RDF/) at Curlie


This page was last edited on 7 January 2019, at 04:30 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation,
Inc., a non-profit organization.
Abstract

This primer is designed to provide the reader with the basic knowledge required to effectively use RDF. It introduces the basic concepts of RDF and shows concrete examples of the use of RDF. Secs. 3-5 can be used as a minimalist introduction into the key elements of RDF. Changes between RDF 1.1 and RDF 1.0 (2004 version) are summarized in a separate document: "What's New in RDF 1.1" [RDF11-NEW].

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is part of the RDF 1.1 document suite. It is an informative note on the key concepts of RDF. For a normative specification of RDF 1.1 the reader is referred to the RDF 1.1. Concepts and Abstract Syntax document [RDF11-CONCEPTS]. This document contains some minor editorial fixes of the February 2014 version.

This document was published by the RDF Working Group as a Working Group Note. If you wish to make comments regarding this document, please send them to public-rdf-comments@w3.org (subscribe, archives). All comments are welcome.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction
2. Why Use RDF?
3. RDF Data Model
   3.1 Triples
   3.2 IRIs
   3.3 Literals
   3.4 Blank nodes
   3.5 Multiple graphs
4. RDF Vocabularies
5. Writing RDF graphs
   5.1 Turtle family of RDF languages
      5.1.1 N-Triples
      5.1.2 Turtle
      5.1.3 Trig
   5.1.4 N-Quads
5.2 JSON-LD
1. Introduction

The Resource Description Framework (RDF) is a framework for expressing information about resources. Resources can be anything, including documents, people, physical objects, and abstract concepts.

RDF is intended for situations in which information on the Web needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

In particular RDF can be used to publish and interlink data on the Web. For example, retrieving http://www.example.org/bob#me could provide data about Bob, including the fact that he knows Alice, as identified by her IRI (an IRI is an "International Resource Identifier"; see Section 3.2 for details). Retrieving Alice’s IRI could then provide more data about her, including links to other datasets for her friends, interests, etc. A person or an automated process can then follow such links and aggregate data about these various things. Such uses of RDF are often qualified as Linked Data [LINKED-DATA].

This document is not normative and does not give a complete account of RDF 1.1. Normative specifications of RDF can be found in the following documents:

- A document describing the basic concepts underlying RDF, as well as abstract syntax ("RDF Concepts and Abstract Syntax") [RDF11-CONCEPTS]
- A document describing the formal model-theoretic semantics of RDF ("RDF Semantics") [RDF11-MT]
- Specifications of serialization formats for RDF:
  - Turtle [TURTLE] and TriG [TRIG]
  - JSON-LD [JSON-LD] (JSON based)
  - RDFa [RDFA-PRIMER] (for HTML embedding)
  - N-Triples [N-TRIPLES] and N-Quads [N-QUADS] (line-based exchange formats)
  - RDF/XML [RDF11-XML] (the original 2004 syntax, updated for RDF 1.1)
- A document describing RDF Schema [RDF11-SCHEMA], which provides a data-modeling vocabulary for RDF data.

2. Why Use RDF?

The following illustrates various different uses of RDF, aimed at different communities of practice.

- Adding machine-readable information to Web pages using, for example, the popular schema.org vocabulary, enabling them to be displayed in an enhanced format on search engines or to be processed automatically by third-party applications.
- Enriching a dataset by linking it to third-party datasets. For example, a dataset about paintings could be enriched by linking them to the corresponding artists in Wikidata, therefore giving access to a wide range of information about them and related resources.
- Interlinking API feeds, making sure that clients can easily discover how to access more information.
- Using the datasets currently published as Linked Data [LINKED-DATA], for example building aggregations of data around specific topics.
- Building distributed social networks by interlinking RDF descriptions of people across multiple Web sites.
- Providing a standards-compliant way for exchanging data between databases.
- Interlinking various datasets within an organisation, enabling cross-dataset queries to be performed using SPARQL [SPARQL11-OVERVIEW].

3. RDF Data Model

3.1 Triples

RDF allows us to make statements about resources. The format of these statements is simple. A statement always has the following structure:

<subject> <predicate> <object>

An RDF statement expresses a relationship between two resources. The subject and the object represent the two resources being related; the predicate represents the nature of their relationship. The relationship is phrased in a directional way (from subject to object) and is called in RDF a property. Because RDF statements consist of three elements they are called triples.

Here are examples of RDF triples (informally expressed in pseudocode):

EXAMPLE 1: Sample triples (informal)

<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde à Washington'> <is about> <the Mona Lisa>
The same resource is often referenced in multiple triples. In the example above, Bob is the subject of four triples, and the Mona Lisa is the subject of one and the object of two triples. This ability to have the same resource be in the subject position of one triple and the object position of another makes it possible to find connections between triples, which is an important part of RDF's power.

We can visualize triples as a connected graph. Graphs consists of nodes and arcs. The subjects and objects of the triples make up the nodes in the graph; the predicates form the arcs. Fig. 1 shows the graph resulting from the sample triples.

![Informal graph of the sample triples](image)

Fig. 1 Informal graph of the sample triples

Once you have a graph like this you can use SPARQL [SPARQL11-OVERVIEW] to query for e.g. people interested in paintings by Leonardo da Vinci.

The RDF Data Model is described in this section in the form of an "abstract syntax", i.e. a data model that is independent of a particular concrete syntax (the syntax used to represent triples stored in text files). Different concrete syntaxes may produce exactly the same graph from the perspective of the abstract syntax. The semantics of RDF graphs [RDF11-MT] are defined in terms of this abstract syntax. Concrete RDF syntax is introduced later in Sec. 5.

In the next three subsections we discuss the three types of RDF data that occur in triples: IRIs, literals and blank nodes.

### 3.2 IRIs

The abbreviation IRI is short for "International Resource Identifier". An IRI identifies a resource. The URLs (Uniform Resource Locators) that people use as Web addresses are one form of IRI. Other forms of IRI provide an identifier for a resource without implying its location or how to access it. The notion of IRI is a generalization of URI (Uniform Resource Identifier), allowing non-ASCII characters to be used in the IRI character string. IRIs are specified in RFC 3987 [RFC3987].

IRIs can appear in all three positions of a triple.

As mentioned, IRIs are used to identify resources such as documents, people, physical objects, and abstract concepts. For example, the IRI for Leonardo da Vinci in DBpedia is:

http://dbpedia.org/resource/Leonardo_da_Vinci

The IRI for an INA video about the Mona Lisa entitled 'La Joconde à Washington' in Europeana is:

http://data.europeana.eu/item/04802/243FA861893BF4117025F17A8B813C5F9AA4D619

IRIs are global identifiers, so other people can re-use this IRI to identify the same thing. For example, the following IRI is used by many people as an RDF property to state an acquaintance relationship between people:

http://xmlns.com/foaf/0.1/knows

RDF is agnostic about what the IRI represents. However, IRIs may be given meaning by particular vocabularies or conventions. For example, DBpedia uses IRIs of the form http://dbpedia.org/resource/Name to denote the thing described by the corresponding Wikipedia article.

### 3.3 Literals

Literals are basic values that are not IRIs. Examples of literals include strings such as "La Joconde", dates such as "the 4th of July, 1990" and numbers such as "3.14159". Literals are associated with a datatype enabling such values to be parsed and interpreted correctly. String literals can optionally be associated with a language tag. For example, "Léonard de Vinci" could be associated with the "fr" language tag and "李奧納多·達文西" with the "zh" language tag.

Literals may only appear in the object position of a triple.

The RDF Concepts document provides a (non-exhaustive) list of datatypes. This includes many datatypes defined by XML Schema, such as string, boolean, integer, decimal and date.
3.4 Blank nodes

IRIs and literals together provide the basic material for writing down RDF statements. In addition, it is sometimes handy to be able to talk about resources without bothering to use a global identifier. For example, we might want to state that the Mona Lisa painting has in its background an unidentified tree which we know to be a cypress tree. A resource without a global identifier, such as the painting's cypress tree, can be represented in RDF by a blank node. Blank nodes are like simple variables in algebra; they represent something without saying what their value is.

Blank nodes can appear in the subject and object position of a triple. They can be used to denote resources without explicitly naming them with an IRI.

Fig. 2 Informal blank node example: the background of the Mona Lisa depicts an unnamed resource that belongs to the class of cypress trees.

3.5 Multiple graphs

RDF provides a mechanism to group RDF statements in multiple graphs and associate such graphs with an IRI. Multiple graphs are a recent extension of the RDF data model. In practice, RDF tool builders and data managers needed a mechanism to talk about subsets of a collection of triples. Multiple graphs were first introduced in the RDF query language SPARQL. The RDF data model was therefore extended with a notion of multiple graphs that is closely aligned with SPARQL.

Multiple graphs in an RDF document constitute an RDF dataset. An RDF dataset may have multiple named graphs and at most one unnamed ("default") graph.

For example, the statements in Example 1 could be grouped in two named graphs. A first graph could be provided by a social networking site and identified by http://example.org/bob:

```
EXAMPLE 2: First graph in the sample dataset

<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
```

The IRI associated with the graph is called the graph name.

A second graph could be provided by Wikidata and identified by https://www.wikidata.org/wiki/Special:EntityData/Q12418:

```
EXAMPLE 3: Second graph in the sample dataset

<Leonardo da Vinci> <is the creator of> <the Mona Lisa>.
<The video 'La Joconde à Washington'> <is about> <the Mona Lisa>
```

Below is an example of an unnamed graph. It contains two triples that have the graph name http://example.org/bob as subject. The triples associate publisher and license information with this graph IRI:

```
EXAMPLE 4: Unnamed graph in the sample dataset

<http://example.org/bob> <has license> <http://creativecommons.org/licenses/by/3.0/>.
```
In this example dataset we assume graph names represent the source of the RDF data held within the corresponding graphs, i.e. by retrieving <http://example.org/bob> we would get access to the four triples in that graph.

**NOTE**

RDF provides no standard way to convey this semantic assumption (i.e., that graph names represent the source of the RDF data) to other readers of the dataset. Those readers will need to rely on out-of-band knowledge, such as established community practice, to interpret the dataset in the intended way. Possible semantics of datasets are described in a separate note [RDF11-DATASETS].

Fig. 3 depicts the sample dataset. Sec. 5.1.3 provides an example of concrete syntax for this dataset.

4. RDF Vocabularies

The RDF data model provides a way to make statements about resources. As we mentioned, this data model does not make any assumptions about what resource IRIs stand for. In practice, RDF is typically used in combination with vocabularies or other conventions that provide semantic information about these resources.

To support the definition of vocabularies RDF provides the RDF Schema language [RDF11-SCHEMA]. This language allows one to define semantic characteristics of RDF data. For example, one can state that the IRI `http://www.example.org/friendOf` can be used as a property and that the subjects and objects of `http://www.example.org/friendOf` triples must be resources of class `http://www.example.org/Person`.

RDF Schema uses the notion of class to specify categories that can be used to classify resources. The relation between an instance and its class is stated through the type property. With RDF Schema one can create hierarchies of classes and sub-classes and of properties and sub-properties. Type restrictions on the subjects and objects of particular triples can be defined through domain and range restrictions. An example of a domain restriction was given above: subjects of "friendOf" triples should be of class "Person".

The main modeling constructs provided by RDF Schema are summarized in the table below:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntactic form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class (a class)</td>
<td><code>rdfs:Class</code></td>
<td>C (a resource) is an RDF class</td>
</tr>
<tr>
<td>Property (a class)</td>
<td><code>rdf:Property</code></td>
<td>P (a resource) is an RDF property</td>
</tr>
<tr>
<td>type (a property)</td>
<td><code>rdfs:Class</code></td>
<td>I (a resource) is an instance of C (a class)</td>
</tr>
<tr>
<td>subClassOf (a property)</td>
<td><code>rdfs:subClassOf</code></td>
<td>C1 (a class) is a subclass of C2 (a class)</td>
</tr>
<tr>
<td>subPropertyOf (a property)</td>
<td><code>rdfs:subPropertyOf</code></td>
<td>P1 (a property) is a sub-property of P2 (a property)</td>
</tr>
</tbody>
</table>
With the help of RDF Schema one can build a model of RDF data. A simple informal example:

**EXAMPLE 5: RDF Schema triples (informal)**

```
<Person> <type> <Class>
<is a friend of> <type> <Property>
<is a friend of> <domain> <Person>
<is a friend of> <range> <Person>
<is a good friend of> <subPropertyOf> <is a friend of>
```

Note that, while `<is a friend of>` is a property typically used as the predicate of a triple (as it was in Example 1), properties like this are themselves resources that can be described by triples or provide values in the descriptions of other resources. In this example, `<is a friend of>` is the subject of triples that assign type, domain, and range values to it, and it is the object of a triple that describes something about the `<is a good friend of>` property.

One of the first RDF vocabularies used worldwide was the "Friend of a Friend" (FOAF) vocabulary for describing social networks. Other examples of RDF vocabularies are:

- **Dublin Core**
  - The Dublin Core Metadata Initiative maintains a metadata element set for describing a wide range of resources. The vocabulary provides properties such as "creator", "publisher" and "title".
- **schema.org**
  - Schema.org is a vocabulary developed by a group of major search providers. The idea is that webmasters can use these terms to mark-up Web pages, so that search engines understand what the pages are about.
- **SKOS**
  - SKOS is a vocabulary for publishing classification schemes such as terminologies and thesauri on the Web. SKOS is since 2009 a W3C recommendation and is widely used in the library world. The Library of Congress published its Subject Headings as a SKOS vocabulary.

Vocabularies get their value from reuse: the more vocabulary IRIs are reused by others, the more valuable it becomes to use the IRIs (the so-called network effect). This means you should prefer re-using someone else's IRI instead of inventing a new one.

For a formal specification of the semantics of the RDF Schema constructs the reader is referred to the RDF Semantics document [RDF11-MT]. Users interested in more comprehensive semantic modeling of RDF data might consider using OWL [OWL2-OVERVIEW]. OWL is an RDF vocabulary, so it can be used in combination with RDF Schema.

### 5. Writing RDF graphs

A number of different serialization formats exist for writing down RDF graphs. However, different ways of writing down the same graph lead to exactly the same triples, and are thus logically equivalent.

In this section we briefly introduce, through annotated examples, the following formats:

1. Turtle family of RDF languages (N-Triples, Turtle, TriG and N-Quads);
2. JSON-LD (JSON-based RDF syntax);
3. RDFa (for HTML and XML embedding);
4. RDF/XML (XML syntax for RDF).

**NOTE**

Reading tip: Sec. 5.1 (Turtle et al.) discusses all basic concepts for serializing RDF. We suggest you read the sections on JSON-LD, RDFa and RDF/XML only if you are interested in that particular usage of RDF.

### 5.1 Turtle family of RDF languages

In this subsection we introduce four RDF languages which are closely related. We start with N-Triples, as it provides basic syntax for writing down RDF triples. The Turtle syntax extends this basic syntax with various forms of syntactic sugar to improve readability. Subsequently we discuss TriG and N-Quads, which are extensions respectively of Turtle and N-Triples to encode multiple graphs. Together, these four are referred to as the "Turtle family of RDF languages".

#### 5.1.1 N-Triples

N-Triples [N-TRIPLES] provides a simple line-based, plain-text way for serializing RDF graphs. The informal graph in Fig. 1 can be represented in N-Triples in the following way:

**EXAMPLE 6: N-Triples**

```
01    <http://example.org/bob#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
02    <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .
```
Each line represents a triple. Full IRIs are enclosed in angle brackets (<>). The period at the end of the line signals the end of the triple. In line 3 we see an example of a literal, in this case a date. The datatype is appended to the literal through a `^^` delimiter. The date representation follows the conventions of the XML Schema datatype date.

Because string literals are so ubiquitous N-Triples allows the user to omit the datatype when writing a string literal. Thus, "Mona Lisa" in line 5 is equivalent to "Mona Lisa"^^xsd:string. In case of language-tagged strings the tag appears directly after the string, separated by a `@` symbol, e.g. "La Joconde"@fr (the French name of the Mona Lisa).

The figure below shows the triples resulting from the example:

![Fig. 4 RDF graph resulting from the N-Triples example](image)

Note that the seven lines in the N-Triples example correspond to the seven arcs in the diagram above.

N-Triples is often used for exchanging large amounts of RDF and for processing large RDF graphs with line-oriented text processing tools.

### 5.1.2 Turtle

Turtle [TURTLE] is an extension of N-Triples. In addition to the basic N-Triples syntax, Turtle introduces a number of syntactic shortcuts, such as support for namespace prefixes, lists and shorthands for datatyped literals. Turtle provides a trade-off between ease of writing, ease of parsing and readability. The graph shown in Fig. 4 can be represented in Turtle as follows:

![EXAMPLE 7: Turtle](image)
The Turtle example is logically equivalent to the N-Triples example. Lines 1-6 contain a number of directives which provide shorthands for writing down IRIs. Relative IRIs (such as `bob#me` on line 8) are resolved against a base IRI, specified here in line 1. Lines 2-6 define IRI prefixes (such as `foaf:`), which can be used for prefixed names (such as `foaf:Person`) instead of full IRIs. The corresponding IRI is constructed by replacing the prefix with its corresponding IRI (in this example `foaf:Person` stands for `<http://xmlns.com/foaf/0.1/Person>`).

Lines 8-12 show how Turtle provides a shorthand for a set of triples with the same subject. Lines 9-11 specify the predicate-object part of triples that have `<http://example.org/bob#me>` as their subject. The semicolons at the end of lines 9-11 indicate that the predicate-object pair that follows it is part of a new triple that uses the most recent subject shown in the data — in this case `bob#me`.

Line 9 shows an example of a special kind of syntactic sugar. The triple should informally be read as "Bob (is) a Person". The `a` predicate is a shorthand for the property `rdf:type` which models the instance relation (see Table 1). The `a` shorthand is intended to match the human intuition about `rdf:type`.

Representation of blank nodes

Below we see two syntactic variants for writing down blank nodes, using the earlier cypress tree example.

**EXAMPLE 8: Blank node**

```
PREFIX lio: <http://purl.org/net/lio#>
<http://dbpedia.org/resource/Mona_Lisa> lio:shows _:x .
_:x a <http://dbpedia.org/resource/Cypress> .
```

The term `_x` is a blank node. It represents an unnamed resource depicted in the Mona Lisa painting; the unnamed resource is an instance of the Cypress class. The example above provides concrete syntax for the informal graph in Fig. 2.

Turtle also has an alternative notation for blank nodes, which does not require the use of syntax like `_x`:

**EXAMPLE 9: Blank nodes (alternative notation)**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
# Some resource (blank node) is interested in some other resource
# entitled "Mona Lisa" and created by Leonardo da Vinci.
[] foaf:topic_interest {
  dcterms:title "Mona Lisa" ;
  dcterms:creator <http://dbpedia.org/resource/Leonardo_da_Vinci> ;
} .
```

Square brackets represent here a blank node. Predicate-object pairs within the square brackets are interpreted as triples with the blank node as subject. Lines starting with `#` represent comments.

For more details about the syntax of Turtle please consult the Turtle specification [TURTLE].

### 5.1.3 TriG

The syntax of Turtle supports only the specification of single graphs without a means for "naming" them. TriG [TRIG] is an extension of Turtle enabling the specification of multiple graphs in the form of an RDF dataset.

**NOTE**

In RDF 1.1 any legal Turtle document is a legal TriG document. One could view it as one language. The names Turtle and TriG still exist for historical reasons.

The multiple-graphs version of our example can be specified in TriG as follows:

**EXAMPLE 10: TriG**

```
01    BASE   <http://example.org/>
02    PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
03    PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
04    PREFIX schema: <http://schema.org/>
05    PREFIX dcterms: <http://purl.org/dc/terms/>
06    PREFIX wd: <http://www.wikidata.org/entity/> .
07
08    GRAPH <http://example.org/bob> {
09        <bob#me>
10            a foaf:Person ;
11            foaf:knows <alice#me> ;
12            schema:birthDate "1990-07-04"^^xsd:date ;
13            foaf:topic_interest wd:Q12418 .
14        }
15
16    GRAPH <http://www.wikidata.org/wiki/Special:EntityData/Q12418> {
17        wd:Q12418
18            dcterms:title "Mona Lisa" ;
20        }
21
22    <http://data.europeana.eu/item/04802/243FA8618938F4117025F17AB8B813C5F9AA4D619>
23        dcterms:subject wd:Q12418 .
```
This RDF dataset contains two named graphs. Lines 8 and 17 list the names of these two graphs. The triples in the named graph are placed in between matching curly braces (lines 9 & 15, 18 & 25). Optionally you can precede the graph name with the keyword `GRAPH`. This may improve readability, but it is mainly introduced for alignment with SPARQL Update [SPARQL11-UPDATE].

The syntax of the triples and of the directives at the top conforms to the Turtle syntax.

The two triples specified on lines 27-29 are not part of any named graph. Together they form the unnamed (“default”) graph of this RDF dataset.

The figure below shows the triples resulting from this example.

![Fig. 5 Triples resulting from the TriG example](image)

### 5.1.4 N-Quads

N-Quads [N-QUADS] is a simple extension to N-Triples to enable the exchange of RDF datasets. N-Quads allows one to add a fourth element to a line, capturing the graph IRI of the triple described on that line. Here is the N-Quads version of the TriG example above:

```
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/type> <http://xmlns.com/foaf/0.1/Person> "Person"
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> "줌공개"
<http://example.org/bob#me> <http://schema.org/birthDate> "1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> "1990-07-04"
<http://example.org/bob#me> <http://xmlns.com/foaf/0.1/topic_interest> <http://www.wikidata.org/entity/Q12418> "Mona Lisa"
```

The nine lines in the N-Quads example correspond to the nine arcs in **Fig. 5**. Lines 1-7 represent quads, where the first element constitutes the graph IRI. The part of the quad after the graph IRI specifies the subject, predicate and object of the statement, following the syntactic
conventions of N-Triples. Lines 8 and 9 represent the statements in the unnamed (default) graph, which lack a fourth element and thus constitute regular triples.

Like N-Triples, N-Quads is typically used for exchanging large RDF datasets and for processing RDF with line-oriented text processing tools.

5.2 JSON-LD

JSON-LD [JSON-LD] provides a JSON syntax for RDF graphs and datasets. JSON-LD can be used to transform JSON documents to RDF with minimal changes. JSON-LD offers universal identifiers for JSON objects, a mechanism in which a JSON document can refer to an object described in another JSON document elsewhere on the Web, as well as datatype and language handling. JSON-LD also provides a way to serialize RDF datasets through the use of the @graph keyword.

The following JSON-LD example encodes the graph of Fig. 4:

```
EXAMPLE 12: JSON-LD
01    {  
02      "@context": 'example-context.json',  
03      "@id": 'http://example.org/bob#me',  
04      "@type": "Person",  
05      "birthdate": '1990-07-04',  
06      "knows": [  
07        {  
08          "@id": 'http://www.wikidata.org/entity/Q12418',  
09          "title": 'Mona Lisa',  
10          "subject_of": 'http://data.europeana.eu/item/04802/2A43FA861938F4117025F17A8B813C5F9AA4D619',  
11          "creator": 'http://dbpedia.org/resource/Leonardo_da_Vinci'  
12        }  
13      ]
14    }
```

The @context key on line 2 points to a JSON document describing how the document can be mapped to an RDF graph (see below). Each JSON object corresponds to an RDF resource. In this example the main resource being described is http://example.org/bob#me, as specified on line 3, through the use of the @id keyword. The @id keyword, when used as a key in a JSON-LD document, points to an IRI identifying the resource corresponding to the current JSON object. We describe the type of this resource on line 4, its birth date on line 5 and one of its friends on line 6. From line 7 to 12 we describe one of its interests, the Mona Lisa painting.

To describe this painting we create a new JSON object on line 7 and associate it with the Mona Lisa IRI in Wikidata on line 8. We then describe various properties of that painting from line 9 to line 11.

The JSON-LD context used in this example is given below.

```
EXAMPLE 13: JSON-LD context specification
01    {  
02      "@context": {  
03        "foaf": "http://xmlns.com/foaf/0.1/",  
04        "Person": "foaf:Person",  
05        "interest": "foaf:topic_interest",  
06        "knows": {  
07          "@id": "foaf:knows",  
08          "@type": "@id"  
09        },  
10        "birthdate": {  
11          "@id": "http://schema.org/birthDate",  
12          "@type": "http://www.w3.org/2001/XMLSchema#date"  
13        },  
14        "dcterms": "http://purl.org/dc/terms/",  
15        "title": "dcterms:title",  
16        "creator": {  
17          "@id": "dcterms:creator",  
18          "@type": "@id"  
19        },  
20        "subject_of": {  
21          "@reverse": "dcterms:subject",  
22          "@type": "@id"  
23        }  
24      }  
25    }
```

This context describes how a JSON-LD document can be mapped to an RDF graph. Lines 4 to 9 specify how to map person, interest and knows to types and properties in the FOAF namespace defined on line 3. We also specify on line 8 that the knows key has a value that will be interpreted as an IRI, through the use of the @type and @id keywords.

From line 10 to line 12 we map birthdate to a schema.org property IRI and specify that its value can be mapped to an xsd:date datatype.

From line 14 to line 23 we describe how to map title, creator and subject_of to Dublin Core property IRIs. The @reverse keyword on line 21 is used to specify that, whenever we encounter "subject_of": "x" in a JSON-LD document using this context, we should map it to an RDF triple which subject is the x IRI, which property is dcterms:subject and which object is the resource corresponding to the parent JSON object.

5.3 RDFa

RDFa [RDFA-PRIMER] is an RDF syntax that can be used to embed RDF data within HTML and XML documents. This enables, for example, search engines to aggregate this data when crawling the Web and use it to enrich search results (see, e.g., schema.org and Rich Snippets).

The HTML example below encodes the RDF graph depicted in Fig. 4:
The RDF/XML example below encodes the RDF graph depicted in Fig. 4.

```xml
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:schema="http://schema.org/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://example.org/bob#me"
  xml:base="http://example.org/bob#me">
  <rdf:Description rdf:about="http://example.org/bob#me">
    <dcterms:isPartOf rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:language>en</dcterms:language>
    <dcterms:title>Mona Lisa</dcterms:title>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
  </rdf:Description>
</rdf:RDF>
```

The example above contains four special RDFa attributes to enable specification of RDF triples within HTML: `resource`, `property`, `typeof` and `prefix`. The `prefix` attribute in line 1 specifies IRI shorthands in a similar fashion as the Turtle prefixes. Strictly speaking, these particular prefixes could have been omitted, as RDFa has a list of predefined prefixes which includes the ones used in this example.

The `div` elements in lines 4 and 14 have a `resource` attribute specifying the IRI about which RDF statements can be made within this HTML element. The meaning of the `typeof` attribute in line 4 is similar to the `is` shorthand in Turtle: the subject `http://example.org/bob#me` is an instance (of the class `foaf:Person`).

In line 6 we see a `property` attribute; the value of this attribute (`foaf:knows`) is interpreted as an RDF property IRI; the value of the `href` attribute (`http://example.org/alice#me`) is interpreted here as the object of the triple. Thus, the RDF statement that results from line 6 is:

```xml
  <a href="http://example.org/alice#me" property="foaf:knows">Alice</a>
```

In line 7 we see a triple with as object a literal value. The `property` attribute is specified here on the HTML `time` element. HTML requires that the content of the time element should be some valid `xsd:date`. By using the built-in HTML semantics of the `time` element RDFa can interpret the value as an `xsd:date` without an explicit datatype declaration.

In lines 10-11 we see the `resource` attribute also being used for specifying the object of a triple. This approach is used when the object is an IRI and the IRI itself is not part of the HTML content (such as an `href` attribute). Line 16 contains a second example of a literal ("Mona Lisa"), defined here as content of the `span` attribute. If RDFa cannot infer the datatype of the literal, it will assume the datatype to be `xsd:string`.

It is not always possible to define RDF statements as part of the HTML content of the document. In that case it is possible to use HTML constructs that do not render content to specify a triple. An example can be found on lines 22-23. The HTML `link` element on line 23 is used here to specify what the subject of the Europeana video (line 22) is.

The use of RDFa in this example is limited to RDFa Lite [RDFA-LITE]. For more information about RDFa please consult the RDFa Primer [RDFA-PRIMER].

### 5.4 RDF/XML

RDF/XML [RDFF-SYNTAX-GRAMMAR] provides an XML syntax for RDF graphs. When RDF was originally developed in the late 1990s, this was its only syntax, and some people still call this syntax "RDF". In 2001, a precursor to Turtle called "N3" was proposed, and gradually the other languages listed here have been adopted and standardized.

The RDF/XML example below encodes the RDF graph depicted in Fig. 4:

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:schema="http://schema.org/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://example.org/bob#me"
  xml:base="http://example.org/bob#me">
  <rdf:Description rdf:about="http://example.org/bob#me">
    <dcterms:isPartOf rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:language>en</dcterms:language>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
  </rdf:Description>
</rdf:RDF>
```
In RDF/XML RDF triples are specified within an XML element `rdf:RDF` (lines 2 and 20). The attributes of the `rdf:RDF` start tag (lines 3-6) provide a shorthand for writing down names of XML elements and attributes. The XML element `rdf:Description` (short for `http://www.w3.org/1999/02/22-rdf-syntax-ns#Description`) is used to define sets of triples that have as subject the IRI specified by the `about` attribute. The first description block (line 7-12) has four sub-elements. The name of the sub-element is an IRI representing an RDF property, e.g., `rdf:type` (line 8). Here, each subelement represents one triple. In cases where the object of the triple is also an IRI the property subelement has no content and the object IRI is specified using the `rdf:resource` attribute (lines 8, 10-11, 15 and 18). For example, line 10 corresponds to the triple:

   <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows> <http://example.org/alice#me> .

When the object of the triple is a literal the literal value is entered as content of the property element (lines 9 and 14). The datatype is specified as attribute of the property element (line 9). If the datatype is omitted (line 14) and no language tag is present the literal is considered to have the datatype `xsd:string`.

The example shows the baseline syntax; please consult the RDF/XML document [RDF11-XML] for a more in-depth treatment of the syntax. It might seem strange that the attribute values contain full IRIs, despite the fact that for some of these namespace prefixes were defined. This is because these prefixes can only be used for XML element and attribute names.

6. Semantics of RDF Graphs

An overarching goal in the use of RDF is to be able to automatically merge useful information from multiple sources to form a larger collection that is still coherent and useful. As a starting point for this merging, all the information is conveyed in the same simple style, subject-predicate-object triples, as described above. To keep the information coherent, however, we need more than just a standard syntax; we also need agreement about the semantics of these triples.

By this point in the Primer, the reader is likely to have an intuitive grasp of the semantics of RDF:

1. The IRIs used to name the subject, predicate, and object are "global" in scope, naming the same thing each time they are used.
2. Each triple is "true" exactly when the predicate relation actually exists between the subject and the object.
3. An RDF graph is "true" exactly when all the triples in it are "true".

These notions, and others, are specified with mathematical precision in the RDF Semantics document [RDF11-MT].

One of the benefits of RDF having these declarative semantics is that systems can make logical inferences. That is, given a certain set of input triples which they accept as true, systems can in some circumstances deduce that other triples must, logically, also be true. We say the first set of triples "entails" the additional triples. These systems, called "reasoners", can also sometimes deduce that the given input triples contradict each other.

Given the flexibility of RDF, where new vocabularies can be created when people want to use new concepts, there are many different kinds of reasoning one might want to do. When a specific kind of reasoning seems to be useful in many different applications, it can be documented as an entailment regime. Several entailment regimes are specified in RDF Semantics. For technical descriptions of some other entailment regimes and how to use these with SPARQL, see [SPARQL11-ENTAILMENT]. Note that some entailment regimes are fairly easy to implement and reasoning can be done quickly, while others require sophisticated techniques to implement efficiently.

As a sample entailment, consider the following two statements:

   ex:bob foaf:knows ex:alice .
   foaf:knows rdfs:domain foaf:Person .

The RDF Semantics document tell us that from this graph it is legal to derive the following triple:

   ex:bob rdf:type foaf:Person .

The derivation above is an example of an RDF Schema entailment [RDF11-MT].

The semantics of RDF also tell us that the triple:

   ex:bob ex:age "forty"^^xsd:integer .

leads to a logical inconsistency, because the literal does not abide by the constraints defined for the XML Schema datatype `integer`.

Note that RDF tools may not recognize all datatypes. As a minimum, tools are required to support the datatypes for string literals and language-tagged literals.

Unlike many other data modeling languages, RDF Schema allows considerable modeling freedom. For example, the same entity may be used as both a class and a property. Also, there is no strict separation between the world of "classes" and of "instances". Therefore, RDF semantics views the following graph as valid:

   ex:Jumbo rdf:type ex:Elephant .
   ex:Elephant rdf:type ex:Species .

So, an elephant can both be a class (with Jumbo as a sample instance) and an instance (namely of the class of animal species).

The examples in this section are just meant to give the reader some feeling about what the RDF Semantics brings you. Please consult [RDF11-MT] for a complete description.

7. RDF Data

RDF allows you to combine triples from any source into a graph and process it as legal RDF. A large amount of RDF data is available as Linked Data [LINKED-DATA]. Datasets are being published and interlinked on the Web using RDF, and many of them offer a querying facility through
SPARQL [SPARQL11-OVERVIEW]. Examples of such datasets used in the examples above include:

- Wikidata, a free, collaborative and multilingual database and ran by the Wikimedia Foundation.
- DBpedia, publishing data extracted from Wikipedia infoboxes.
- WordNet, a lexical database of English terms, grouped in sets of synonyms, with a range of semantic interrelations. Similar databases exist for other languages.
- Europeana, publishing data about cultural objects from a large number of European institutions.
- VIAF, publishing data about people, works and geographic places from a number of national libraries and other agencies.

A list of datasets available as Linked Data is maintained at datahub.io.

A number of vocabulary terms have become popular for recording links between RDF data sources. An example is the `sameAs` property provided by the OWL vocabulary. This property can be used to indicate that two IRIs point in fact to the same resource. This is useful because different publishers may use different identifiers to denote the same thing. For example, VIAF (see above) also has an IRI denoting Leonardo da Vinci. With the help of `owl:sameAs` we can record this information:

```
EXAMPLE 16: Link between datasets
```

Such links can be deployed by RDF data-processing software, for example by merging or comparing RDF data of IRIs that point to the same resource.

8. More Information

This concludes our brief introduction into RDF. Please consult the references to get more detailed information. You might also want to take a look at the W3C Linked Data page.

A. Acknowledgments

Antoine Isaac provided many examples, including the different syntactic forms. Pierre-Antoine Champin provided one of the JSON-LD examples. Andrew Wood designed the diagrams. Sandro Hawke wrote the first part of the section on RDF semantics.

We are grateful for the comments provided by (in alphabetical order) Gareth Adams, Thomas Baker, Dan Brickley, Pierre-Antoine Champin, Bob DuCharme, Sandro Hawke, Patrick Hayes, Ivan Herman, Kingsley Idehen, Antoine Isaac, Markus Lanthaler, and David Wood.

The introduction of this document contains a number of sentences from the 2004 Primer [RDF-PRIMER]. For the rest the RDF 1.1 Primer is a completely new document.

B. Changes since the previous publication

- Small editorial issues fixed (including three listed errata).
- Link to Japanese translation added.

C. References

C.1 Informative references

[JSON-LD]
Manu Sporny, Gregg Kellogg, Markus Lanthaler, Editors. JSON-LD 1.0. 16 January 2014. W3C Recommendation. URL: http://www.w3.org/TR/json-ld/

[LINKED-DATA]
Tim Berners-Lee. Linked Data. Personal View, imperfect but published. URL: http://www.w3.org/DesignIssues/LinkedData.html

[N-QUADS]

[N-TRIPLES]

[OWL2-OVERVIEW]

[RDF-PRIMER]
Frank Manola; Eric Miller. RDF Primer. 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/rdf-primer/

[RDF-SYNTAX-GRAMMAR]
Fabien Gandon; Guus Schreiber. RDF 1.1 XML Syntax. 25 February 2014. W3C Recommendation. URL: http://www.w3.org/TR/rdf-syntax-grammar/

[RDF11-CONCEPTS]

[RDF11-DATASETS]

[RDF11-NEW]

[RDF11-SCHEMA]

[RDF11-XML]

[RDF11-LITE]
Manu Sporny. *RDFa Lite 1.1*. 7 June 2012. W3C Recommendation. URL: http://www.w3.org/TR/rdfa-lite/  

[RDF11-PRIMER]
Ivan Herman; Ben Adida; Manu Sporny; Mark Birbeck. *RDFa 1.1 Primer - Second Edition*. 22 August 2013. W3C Note. URL: http://www.w3.org/TR/rdfa-primer/  

[RFC3987]

[SPARQL11-ENTAILMENT]
Birte Glimm; Chimezie Ogbuji. *SPARQL 1.1 Entailment Regimes*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-entailment/  

[SPARQL11-OVERVIEW]

[SPARQL11-UPDATE]
Paul Gearon; Alexandre Passant; Axel Polleres. *SPARQL 1.1 Update*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-update/  

[TRIG]

[TURTLE]
IIIF Authentication API 1.0

Status of this Document

This Version: 1.0.0
Latest Stable Version: 1.0.0 (/api/auth/1.0/)
Previous Version: 0.9.4 (/api/auth/0.9/)

Editors:

Michael Appleby (https://orcid.org/0000-0002-1266-298X) (https://orcid.org/0000-0002-1266-298X), Yale University (http://www.yale.edu/)
Tom Crane (https://orcid.org/0000-0003-1881-243X) (https://orcid.org/0000-0003-1881-243X), Digirati (http://digirati.com/)
Robert Sanderson (https://orcid.org/0000-0003-4441-6852) (https://orcid.org/0000-0003-4441-6852), J. Paul Getty Trust (http://www.getty.edu/)
Jon Stroop (https://orcid.org/0000-0002-0367-1243) (https://orcid.org/0000-0002-0367-1243), Princeton University Library (https://library.princeton.edu/)
Simeon Warner (https://orcid.org/0000-0002-7970-7855) (https://orcid.org/0000-0002-7970-7855), Cornell University (https://www.cornell.edu/)

Copyright © 2015-2019 Editors and contributors. Published by the IIIF (International Image Interoperability Framework) Consortium under the CC-BY (http://creativecommons.org/licenses/by/4.0/) license, see disclaimer (/api/annex/notes/disclaimer/).

Table of Contents

1. Introduction
   1.1. Terminology
1.2. Authentication for Content Resources
1.3. Authentication for Description Resources
1. Introduction

The IIIF (International Image Interoperability Framework) (pronounced “Triple-Eye-Eff”) specifications are designed to support uniform and rich access to resources hosted around the world. Open access to content is desirable, but internal policies, legal regulations, business models, and other constraints can require users to authenticate and be authorized to interact with some resources. The authentication process could range from a simple restriction by IP (Internet Protocol) address or a click-through agreement, to a multi-factor scheme with a secure identity provider.

Content providers that need to restrict access to their resources may offer tiered access to alternative versions that go beyond a simple all-or-nothing proposition. These alternative versions could be degraded based on resolution, watermarking, or compression, for example, but are often better than no access at all.

Providing interoperable access to restricted content through client applications running in a web browser poses many challenges:
A single IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) manifest can reference content resources at multiple institutions and hence from multiple domains.

Each version of a resource must have a distinct URI to prevent web caches from providing the wrong version.

Institutions have different existing access control systems.

Most IIIF (International Image Interoperability Framework) viewers are client-side JavaScript applications, and may be served from a domain that is different from, and thus untrusted by, the image services that it is required to load.

Similarly, the domain of the authentication services may be different from that of a viewer or the IIIF (International Image Interoperability Framework)-based content. Therefore, the authorizing server must not require any prior knowledge of the domain hosting the viewer.

Additionally, the IIIF (International Image Interoperability Framework) community has the following goals for this specification:

- A IIIF (International Image Interoperability Framework) client should not authenticate the user itself; the server hosting the content must be responsible for capturing credentials from a user and the IIIF (International Image Interoperability Framework) viewer needs no knowledge of or access to this exchange.
- A browser-based IIIF (International Image Interoperability Framework) client must be able to maintain its own internal state during an authentication flow.
- A registry of trusted domains should not be required; anyone should be able to create any kind of viewer and run it from anywhere.
- Institutions should be able to use their existing authentication systems without modification.

To meet these challenges and goals, the IIIF (International Image Interoperability Framework) Authentication specification describes a set of workflows for guiding the user through an existing access control system. The process of authenticating the user is mostly outside the scope of the specification and may involve a round-trip to a CAS (Central Authentication Service) server, or an OAuth2 provider, or a bespoke login system. In this sense, IIIF (International Image Interoperability Framework) Authentication is not the same as a protocol like CAS (Central Authentication Service); it is a pattern for interacting with arbitrary third party protocols.

IIIF (International Image Interoperability Framework) Authentication provides a link to a user interface for logging in, and services that provide credentials, modeled after elements of the OAuth2 workflow. Together they act as a bridge to the access control system in use on the server, without the client requiring knowledge of that system.

In summary, the specification describes how to:

- From within a viewer, initiate an interaction with an access control system so that a user can acquire the credentials they need to view restricted content.
- Give the client just enough knowledge of the user’s state with respect to the content provider to ensure a good user experience.

Please send feedback to iiif-discuss@googlegroups.com.

1.1. Terminology

This specification distinguishes between Content Resources, such as images or videos, and Description Resources which conform to IIIF (International Image Interoperability Framework) specifications, such as Image API (Application Programming Interface) (/api/image/) image information (info.json) and Presentation API (Application Programming Interface) (/api/presentation/) collection or manifest resources. From the point of view of a browser-based application, Content Resources are
Two additional concepts, the **access cookie** and **access token**, are described below.

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in [RFC 2119](http://tools.ietf.org/html/rfc2119).

### 1.2. Authentication for Content Resources

Content Resources, such as images, are generally secondary resources embedded in a web page or application. In the case of web pages, images might be included via the **HTML** `img` tag, and retrieved via additional **HTTP** (Hypertext Transfer Protocol) requests by the browser. When a user is not authorized to load a web page, the server can redirect the user to another page and offer the opportunity to authenticate. This redirection is not possible for embedded Content Resources, and the user is simply presented with a broken image icon. If the image is access controlled, the browser must avoid broken images by sending a cookie that the server can accept as a credential that grants access to the image. This specification describes the process by which the user acquires this **access cookie**.

### 1.3. Authentication for Description Resources

Description Resources, such as a Presentation **API** (Application Programming Interface) manifest or an **Image API** (Application Programming Interface) information document (info.json), give the client application the information it needs to have the browser request the Content Resources. A Description Resource must be on the same domain as the Content Resource it describes, but there is no requirement that the executing client code is also hosted on this domain.

A browser running JavaScript retrieved from one domain cannot use **XMLHttpRequest** to load a Description Resource from another domain and include that domain's cookies in the request, without violating the requirement introduced above that the client must work when **untrusted**. Instead, the client sends an **access token**, technically a type of bearer token ([https://tools.ietf.org/html/rfc6750#section-1.2](https://tools.ietf.org/html/rfc6750#section-1.2)), as a proxy for the access cookie. This specification describes how, once the browser has acquired the access cookie for the Content Resources, the client acquires the access token to use when making direct requests for Description Resources.

The server on the Resource Domain treats the access token as a representation of, or proxy for, the cookie that grants access to the Content Resources. When the client makes requests for the Description Resources and presents the access token, the responses tell the client what will happen when the browser requests the corresponding content resources with the access cookie the access token represents. These responses let the client decide what user interface and/or Content Resources to show to the user.

### 1.4. Security

The purpose of this specification to support access-control for **IIIF** (International Image Interoperability Framework) resources and hence security is a core concern. To prevent misuse, cookies and bearer tokens described in this specification need to be protected from disclosure in storage and in transport. Implementations **SHOULD** use **HTTP** (Hypertext Transfer Protocol) over **TLS** (Transport Layer Security) ([https://tools.ietf.org/html/rfc2818](https://tools.ietf.org/html/rfc2818)), commonly known as **HTTPS** (Hypertext Transfer Protocol (Secure)), for all communication. Furthermore, all **IIIF** (International Image Interoperability Framework) clients that interact with access-controlled resources **SHOULD** also be run from pages served via **HTTPS** (Hypertext Transfer Protocol (Secure)). All references to **HTTP** (Hypertext Transfer Protocol) in this specification should be read assuming the use of **HTTPS** (Hypertext Transfer Protocol (Secure)).
This specification protects Content Resources such as images by making the access token value available to the script of the client application, for use in requesting Description Resources. Knowledge of the access token is of no value to a malicious client, because the access cookie (which the client cannot see) is the only credential accepted for Content Resources, and a Description Resource is of no value on its own. However, the interaction patterns introduced in this specification will in future versions be extended to write operations on IIIF (International Image Interoperability Framework) resources, for example creating annotations in an annotation server, or modifying the structures element in a manifest. For these kinds of operations, the access token is the credential, and the flow introduced below may require one or more additional steps to establish trust between client and server. However, it is anticipated that these changes will be backwards compatible with version 1.0.

Further discussion of security considerations can be found in the Implementation Notes (implementation/).

2. Authentication Services

Authentication services follow the pattern described in the IIIF (International Image Interoperability Framework) Linking to External Services (/api/annex/services/) note, and are referenced in one or more service blocks from the descriptions of the resources that are protected. There is a primary login service profile for authenticating users, and it has related services nested within its description. The related services include a mandatory access token service, and an optional logout service.

2.1. Access Cookie Service

The client uses this service to obtain a cookie that will be used when interacting with content such as images, and with the access token service. There are several different interaction patterns in which the client will use this service, based on the user interface that must be rendered for the user, indicated by a profile URI. The client obtains the link to the access cookie service from a service block in a description of the protected resource.

The purpose of the access cookie service is to set a cookie during the user’s interaction with the content server, so that when the client then makes image requests to the content server, the requests will succeed. The client has no knowledge of what happens at the login service, and it cannot see any cookies set for the content domain during the user’s interaction with the login service. The browser may be redirected one or more times but this is invisible to the client application. The final response in the opened tab SHOULD contain JavaScript that will attempt to close the tab, in order to trigger the next step in the workflow.

2.1.1. Service Description

There are four interaction patterns by which the client can obtain an access cookie, each identified by a profile URI. These patterns are described in more detail in the following sections.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Profile URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td><a href="http://iiif.io/api/auth/1/login">http://iiif.io/api/auth/1/login</a></td>
<td>The user will be required to log in using a separate window with a UI provided by an external authentication system.</td>
</tr>
<tr>
<td>Clickthrough</td>
<td><a href="http://iiif.io/api/auth/1/clickthrough">http://iiif.io/api/auth/1/clickthrough</a></td>
<td>The user will be required to click a button within the client using content provided in the service description.</td>
</tr>
<tr>
<td>Kiosk</td>
<td><a href="http://iiif.io/api/auth/1/kiosk">http://iiif.io/api/auth/1/kiosk</a></td>
<td>The user will not be required to interact with an authentication system, the client is expected to use the access cookie service automatically.</td>
</tr>
</tbody>
</table>
## Pattern Profile URI Description

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Profile URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td><a href="http://iiif.io/api/auth/1/external">http://iiif.io/api/auth/1/external</a></td>
<td>The user is expected to have already acquired the appropriate cookie, and the access cookie service will not be used at all.</td>
</tr>
</tbody>
</table>

The service description is included in the Description Resource and has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@id</td>
<td>see description</td>
<td>It is REQUIRED with the Login, Clickthrough, or Kiosk patterns, in which the client opens the URI in order to obtain an access cookie. It is OPTIONAL with the External pattern, as the user is expected to have obtained the cookie by other means and any value provided is ignored.</td>
</tr>
<tr>
<td>profile</td>
<td>REQUIRED</td>
<td>The profile for the service MUST be one of the profile URIs from the table above.</td>
</tr>
<tr>
<td>label</td>
<td>REQUIRED</td>
<td>The text to be shown to the user to initiate the loading of the authentication service when there are multiple services required. The value MUST include the domain or institution to which the user is authenticating.</td>
</tr>
<tr>
<td>confirmLabel</td>
<td>RECOMMENDED</td>
<td>The text to be shown to the user on the button or element that triggers opening of the access cookie service. If not present, the client supplies text appropriate to the interaction pattern if needed.</td>
</tr>
<tr>
<td>header</td>
<td>RECOMMENDED</td>
<td>A short text that, if present, MUST be shown to the user as a header for the description, or alone if no description is given.</td>
</tr>
<tr>
<td>description</td>
<td>RECOMMENDED</td>
<td>Text that, if present, MUST be shown to the user before opening the access cookie service.</td>
</tr>
<tr>
<td>failureHeader</td>
<td>OPTIONAL</td>
<td>A short text that, if present, MAY be shown to the user as a header after failing to receive a token, or using the token results in an error.</td>
</tr>
<tr>
<td>failureDescription</td>
<td>OPTIONAL</td>
<td>Text that, if present, MAY be shown to the user after failing to receive a token, or using the token results in an error.</td>
</tr>
</tbody>
</table>

### 2.1.2. Interaction with the Access Cookie Service

The client MUST append the following query parameter to all requests to an access cookie service URI, regardless of the interaction pattern, and open this URI in a new window or tab.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
For example, given an access cookie service URI of https://authentication.example.org/login, a client instantiated by the page https://client.example.com/viewer/index.html would make its request to:


The server MAY use this information to validate the origin supplied in subsequent requests to the access token service, for example by encoding it in the cookie returned.

2.1.3. Login Interaction Pattern

In order to have the client prompt the user to log in, it must display part of the content provider’s user interface. For the Login interaction pattern, the value of the @id property is the URI of that user interface.

The interaction has the following steps:

- If the header and/or description properties are present, before opening the provider’s authentication interface, the client SHOULD display the values of the properties to the user. The properties will describe what is about to happen when they click the element with the confirmLabel.
- When the confirmLabel element is activated, the client MUST then open the URI from @id with the added origin query parameter. This MUST be done in a new window or tab to help prevent spoofing attacks. Browser security rules prevent the client from knowing what is happening in the new tab, therefore the client can only wait for and detect the closing of the opened tab.
- After the opened tab is closed, the client MUST then use the related access token service, as described below.

With out-of-band knowledge, authorized non-user-driven clients MAY use POST to send the pre-authenticated user’s information to the service. As the information required depends on authorization logic, the details are not specified by this API (Application Programming Interface).

An example service description for the Login interaction pattern:

```json
{
    "service": {
        "@context": "http://iiif.io/api/auth/1/context.json",
        "@id": "https://authentication.example.org/login",
        "profile": "http://iiif.io/api/auth/1/login",
        "label": "Login to Example Institution",
        "header": "Please Log In",
        "description": "Example Institution requires that you log in with your example account to view this content."
    },
    "confirmLabel": "Login",
    "failureHeader": "Authentication Failed",
    "failureDescription": "<a href="http://example.org/policy">Access Policy</a>"
}
```

2.1.4. Clickthrough Interaction Pattern

For the Clickthrough interaction pattern, the value of the @id property is the URI of a service that MUST set an access cookie and then immediately close its window or tab without user interaction. The interaction has the following steps:
- If the header and/or description properties are present, before opening the service, the client MUST display the values of the properties to the user. The properties will describe the agreement implied by clicking the element with the confirmLabel.

- When the confirmLabel element is activated, the client MUST then open the URI from @id with the added origin query parameter. This SHOULD be done in a new window or tab. Browser security rules prevent the client from knowing what is happening in the new tab, therefore the client can only wait for and detect the closing of the opened window or tab or iframe.

- After the opened tab is closed, the client MUST then use the related access token service, as described below.

Non-user-driven clients MUST not use access cookie services with the Clickthrough interaction pattern, and instead halt.

An example service description for the Clickthrough interaction pattern:

```json
{
    // ...
    "service": {
        "@context": "http://iiif.io/api/auth/1/context.json",
        "@id": "https://authentication.example.org/clickthrough",
        "profile": "http://iiif.io/api/auth/1/clickthrough",
        "label": "Terms of Use for Example Institution",
        "header": "Restricted Material with Terms of Use",
        "description": "<span>... terms of use ... </span>",
        "confirmLabel": "I Agree",
        "failureHeader": "Terms of Use Not Accepted",
        "failureDescription": "You must accept the terms of use to see the content.",
        "service": {
            // Access token service ...
        }
    }
}
```

### 2.1.5. Kiosk Interaction Pattern

For the Kiosk interaction pattern, the value of the @id property is the URI of a service that MUST set an access cookie and then immediately close its window or tab without user interaction. The interaction has the following steps:

- There is no user interaction before opening the access cookie service URI, and therefore any of the label, header, description and confirmLabel properties are ignored if present.

- The client MUST immediately open the URI from @id with the added origin query parameter. This SHOULD be done in a new window or tab. Browser security rules prevent the client from knowing what is happening in the new tab, therefore the client can only wait for and detect the closing of the opened window or tab or frame.

- After the opened tab is closed, the client MUST then use the related access token service, as described below.

Non-user-driven clients simply access the URI from @id to obtain the access cookie, and then use the related access token service, as described below.

An example service description for the Kiosk interaction pattern:
2.1.6. External Interaction Pattern

For the External interaction pattern, the user is required to have acquired the access cookie by out of band means. If the access cookie is not present, the user will receive the failure messages. The interaction has the following steps:

- There is no user interaction before opening the access token service URI, and therefore any of the label, header, description and confirmLabel properties are ignored if present.
- There is no access cookie service. Any URI specified in the @id property MUST be ignored.
- The client MUST immediately use the related access token service, as described below.

Non-user-driven clients simply use the related access token service with a previously acquired access cookie, as described below.

An example service description for the External interaction pattern:

```json
{
  // ...
  "service": {
    "@context": "http://iiif.io/api/auth/1/context.json",
    "@id": "https://authentication.example.org/cookiebaker",
    "profile": "http://iiif.io/api/auth/1/kiosk",
    "label": "Internal cookie granting service",
    "failureHeader": "Ooops!",
    "failureDescription": "Call Bob at ext. 1234 to reboot the cookie server",
    "service": {
      // Access token service ...
    }
  }
}
```

2.2. Access Token Service

The client uses this service to obtain an access token which it then uses when requesting Description Resources. A request to the access token service must include any cookies for the content domain acquired from the user’s interaction with the corresponding access cookie service, so that the server can issue the access token.

2.2.1. Service Description

The access cookie service description MUST include an access token service description following the template below:
The @id property of the access token service MUST be present, and its value MUST be the URI from which the client can obtain the access token. The profile property MUST be present and its value MUST be http://iiif.io/api/auth/1/token to distinguish it from other services. There is no requirement to repeat the @context property included in the enclosing access cookie service description, and there are no other properties for this service.

2.2.2. The JSON (JavaScript Object Notation) Access Token Response

If the request has a valid cookie that the server recognises as having been issued by the access cookie service, the access token service response MUST include a JSON (JavaScript Object Notation) (not JSON-LD (JSON for Linking Data)) object with the following structure:

```json
{
  "accessToken": "TOKEN_HERE",
  "expiresIn": 3600
}
```

The accessToken property is REQUIRED, and its value is the access token to be passed back in future requests. The expiresIn property is OPTIONAL and, if present, the value is the number of seconds in which the access token will cease to be valid.

Once obtained, the access token MUST be passed back to the server on all future requests for Description Resources by adding an Authorization request header, with the value Bearer followed by a space and the access token, such as:

```
Authorization: Bearer TOKEN_HERE
```

This authorization header SHOULD be added to all requests for resources from the same domain and subdomains that have a reference to the service, regardless of which API (Application Programming Interface) is being interacted with. It MUST NOT be sent to other domains.

2.2.3. Interaction for Non-Browser Client Applications

The simplest access token request comes from a non-browser client that can send cookies across domains, where the CORS (Cross-Origin Resource Sharing) restrictions do not apply. An example URL:

```
https://authentication.example.org/token
```

Would result in the HTTP (Hypertext Transfer Protocol) Request:
GET /token HTTP/1.1
Cookie: <cookie-acquired-during-login>

The response is the JSON (JavaScript Object Notation) access token object with the media type application/json:

```json

{
  "accessToken": "TOKEN_HERE",
  "expiresIn": 3600
}

```

### 2.2.4. Interaction for Browser-Based Client Applications

If the client is a JavaScript application running in a web browser, it needs to make a direct request for the access token and store the result. The client can't use XMLHttpRequest because it can't include the access cookie in a cross-domain request. Instead, the client MUST open the access token service in a frame using an iframe element and be ready to receive a message posted by script in that frame using the postMessage API (Application Programming Interface) (https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage). To trigger this behavior, the client MUST append the following query parameters to the access token service URI, and open this new URI in the frame.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageId</td>
<td>A string that both prompts the server to respond with a web page instead of JSON (JavaScript Object Notation), and allows the client to match access token service requests with the messages received. If a client has no need to interact with multiple token services, it can use a dummy value for the parameter, e.g., messageId=1.</td>
</tr>
</tbody>
</table>

For example, a client instantiated by the page at https://client.example.com/viewer/index.html would request:

```
https://authentication.example.org/token?messageId=1&origin=https://client.example.com/
```

When the server receives a request for the access token service with the messageId parameter, it MUST respond with an HTML (HyperText Markup Language) web page rather than raw JSON (JavaScript Object Notation). The web page MUST contain script that sends a message to the opening page using the postMessage API (Application Programming Interface). The message body is the JSON (JavaScript Object Notation) access token object, with the value of the supplied messageId as an extra property, as shown in the examples in the next section.

The server MAY use the origin information for further authorization logic, even though the user is already authenticated. For example, the server may trust only specific domains for certain actions like creating or deleting resources compared to simply reading them. If the client sends an incorrect value, it will not receive the posted response, as the postMessage API (Application Programming Interface) will not dispatch the event. The targetOrigin parameter of the postMessage() function call MUST be the origin provided in the request.

The frame SHOULD NOT be shown to the user. It is a mechanism for cross-domain messaging. The client MUST register an event listener to receive the message that the token service page in the frame will send. The client can reuse the same listener and frame for multiple calls to the access token service, or it can create new ones for each invocation.

The exact implementation will vary but MUST include features equivalent to the following steps.
The client must first register an event listener to receive a cross domain message:

```javascript
window.addEventListener("message", receive_message);

function receive_message(event) {
  data = event.data;
  var token, error;
  if (data.hasOwnProperty('accessToken')) {
    token = data.accessToken;
  } else {
    // handle error condition
  }
  // ...
}
```

It can then open the access token service in a frame:

```javascript
document.getElementById('messageFrame').src = 'https://authentication.example.org/token?' + 'messageId=1234&origin=https://client.example.com/';
```

The server response will then be a web page with a media type of text/html that can post a message to the registered listener:

```html
<html>
<body>
<script>
  window.parent.postMessage({
    "messageId": "1234",
    "accessToken": "TOKEN_HERE",
    "expiresIn": 3600
  }, 'https://client.example.com/');
</script>
</body>
</html>
```

2.2.5. Using the Access Token

The access token is sent on all subsequent requests for Description Resources. For example, a request for the image information in the Image API (Application Programming Interface) would look like:

```
GET /iiif/identifier/info.json HTTP/1.1
Authorization: Bearer TOKEN_HERE
```

2.2.6. Access Token Error Conditions

The response from the access token service may be an error. The error MUST be supplied as JSON (JavaScript Object Notation) with the following template. For browser-based clients using the `postMessage` API (Application Programming Interface), the error object must be sent to the client via JavaScript, in the same way the access token is sent. For direct requests the response body is the raw JSON (JavaScript Object Notation).
The value of the `error` property MUST be one of the types in the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalidRequest</td>
<td>The service could not process the information sent in the body of the request.</td>
</tr>
<tr>
<td>missingCredentials</td>
<td>The request did not have the credentials required.</td>
</tr>
<tr>
<td>invalidCredentials</td>
<td>The request had credentials that are not valid for the service.</td>
</tr>
<tr>
<td>invalidOrigin</td>
<td>The request came from a different origin than that specified in the access cookie service request, or an origin that the server rejects for other reasons.</td>
</tr>
<tr>
<td>unavailable</td>
<td>The request could not be fulfilled for reasons other than those listed above, such as scheduled maintenance.</td>
</tr>
</tbody>
</table>

The `description` property is OPTIONAL and may give additional human-readable information about the error.

When returning JSON (JavaScript Object Notation) directly, the service MUST use the appropriate HTTP (Hypertext Transfer Protocol) status code for the response to describe the error (for example 400, 401 or 503). The postMessage web page response MUST use the 200 HTTP (Hypertext Transfer Protocol) status code to ensure that the body is received by the client correctly.

2.3. Logout Service

In the case of the Login interaction pattern, the client will need to know if and where the user can go to log out. For example, the user may wish to close their session on a public terminal, or to log in again with a different account.

2.3.1. Service Description

If the authentication system supports users intentionally logging out, there SHOULD be a logout service associated with the access cookie service following the template below:
The value of the `profile` property MUST be `http://iiif.io/api/auth/1/logout`.

2.3.2. Interaction

The client SHOULD present the results of an HTTP (Hypertext Transfer Protocol) GET request on the service’s URI in a separate tab or window with a URL bar. At the same time, the client SHOULD discard any access token that it has received from the corresponding service. The server SHOULD reset the user’s logged in status when this request is made and delete the access cookie.

2.4. Example Description Resource with Authentication Services

The example below is a complete image information response for an example image with all of the authentication services.
3. Interaction with Access-Controlled Resources

This section describes how clients use the services above when interacting with Content Resources and Description Resources.

These interactions rely on requests for Description Resources returning HTTP (Hypertext Transfer Protocol) status codes 200, 302, and 401 in different circumstances. In cases other than 302, the body of the response MUST be a valid Description Resource because the client needs to see the Authentication service descriptions in order to follow the appropriate workflow. Any response with a 302 status code will never be seen by browser-based client script interacting via the XMLHttpRequest API (Application Programming Interface). The reported response will be the final one in the chain, and therefore the body of redirection responses is not required to be the Description Resource’s representation.

3.1. All or Nothing Access

If the server does not support multiple tiers of access to a Content Resource, and the user is not authorized to access it, then the server MUST return a response with a 401 (Unauthorized) HTTP (Hypertext Transfer Protocol) status code for the corresponding Description Resource.
If the user is authorized for a Description Resource, the client can assume that requests for the described Content Resources will also be authorized. Requests for the Content Resources rely on the access cookie to convey the authorization state.

3.2. Tiered Access

If a server supports multiple tiers of access, then it MUST use a different identifier for each Description Resource and its corresponding Content Resource(s). For example, there MUST be different Image Information documents (/info.json) at different URIs for each tier. When referring to Description Resources that have multiple tiers of access, systems SHOULD use the identifier of the version that an appropriated authorized user should see. For example, when referring to an Image service from a Manifest, the reference would normally be to the highest quality image version rather than a degraded version.

When a Description Resource is requested and the user is not authorized to access it and there are lower tiers available, the server MUST issue a 302 (Found) HTTP (Hypertext Transfer Protocol) status response to redirect to the Description Resource of a lower tier.

When there are no lower tiers and the user is not authorized to access the current Description Resource, the server MUST issue a 401 (Unauthorized) response. The client SHOULD present information about the Login and/or Clickthrough services included in the Description Resource to allow the user to attempt to authenticate.

4. Workflow from the Browser Client Perspective

![Client Authentication Workflow](image)

Browser-based clients will perform the following workflow in order to access access controlled resources:

- The client requests the Description Resource and checks the status code of the response.
- If the response is a 200,
  - The client checks whether the @id property in the response is the same URI as was requested.
  - If it is, then the client can proceed to request the Content Resource.
If the URIs are different, then the resource is from a different tier from the requested one. The 200 status implies that the resource is available to be used, and the client can therefore render the resource. At the same time, the client checks for authentication services in the JSON (JavaScript Object Notation) received.

If the response is a 401,

- The client does not have access to the Content Resource, and thus the client checks for authentication services in the JSON (JavaScript Object Notation) received.

If the response is neither 200 nor 401, the client must handle other HTTP (Hypertext Transfer Protocol) status codes.

When the client checks for authentication services, it first checks the authentication services:

- First it looks for an External interaction pattern as this does not require any user interaction. If present, it opens the Access Token service to see if the cookie has already been obtained.
- If not, it checks for a Kiosk interaction pattern as it does not involve user interaction. If present, it opens the Access Cookie service in a separate window.
- If not, it checks for a Clickthrough interaction pattern. If present, it renders the description of the service and a confirmation button to prompt the user to click through. Once the user has clicked the confirmation, it opens the Access Cookie service in a separate window.
- If not, it presents any Login interaction patterns available and prompts the user to login with one of them. When the user selects the realm to log in at, which takes the Access Cookie service role, it opens that realm’s user interface in a separate window.
- When the Access Cookie service window closes, either automatically or by the user, the client opens the Access Token Service.

After the Access Token service has been requested, if the client receives a token, it tries again to read the Description Resource with the newly acquired credentials.

- If the client instead receives an error, it returns to look for further authentication services to interact with.
- If there are no further authentication services, then the user does not have the credentials to interact with any of the Content Resource versions, and the client cannot display anything.

Please note that the server implementation involves providing 302 status responses to redirect the client from the requested tier to another tier if the user is not yet authorized to see the resource. The browser-based client does not see these responses, and hence tests whether the identifier of the resource is the same as the one requested, rather than for the HTTP (Hypertext Transfer Protocol) status code.

Appendices

A. Implementation Notes

Guidance for implementers is provided in a separate Implementation Notes (implementation/) document. The notes cover many details relating to implementation of this specification in browser-based JavaScript applications, and additional security considerations.

B. Versioning
Starting with version 0.9.0, this specification follows Semantic Versioning (http://semver.org/spec/v2.0.0.html). See the note Versioning of APIs (/api/annex/notes/semver/) for details regarding how this is implemented.

C. Acknowledgments

The production of this document was generously supported by a grant from the Andrew W. Mellon Foundation (http://www.mellon.org/).

Many thanks to the members of the IIIF (International Image Interoperability Framework) Community (https://iiif.io/community/) for their continuous engagement, innovative ideas and feedback.

D. Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-01-19</td>
<td>Version 1.0 (Alchemical Key)</td>
</tr>
<tr>
<td>2016-10-05</td>
<td>Version 0.9.4 (Incrementing Integer) add to security notes</td>
</tr>
<tr>
<td>2016-08-22</td>
<td>Version 0.9.3 (Wasabi KitKat) separate profiles, remove client identity service, add query parameters</td>
</tr>
<tr>
<td>(unreleased)</td>
<td>Version 0.9.2 (unnamed) postMessage instead of JSONP (JavaScript Object Notation with Padding)</td>
</tr>
<tr>
<td>2015-10-30</td>
<td>Version 0.9.1 (Table Flip) add missing @context, clarifications</td>
</tr>
<tr>
<td>2015-07-28</td>
<td>Version 0.9.0 (unnamed) draft</td>
</tr>
</tbody>
</table>

Feedback: iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com) Get involved: Join IIIF (https://iiif.io/community/#how-to-get-involved)
IIIF Image API 2.1.1

Status of this Document

This Version: 2.1.1

Latest Stable Version: 2.1.1 (https://iiif.io/api/image/2.1/)

Previous Version: 2.0 (https://iiif.io/api/image/2.0/)

Editors:

- Michael Appleby (https://orcid.org/0000-0002-1266-298X) (https://orcid.org/0000-0002-1266-298X), Yale University (http://www.yale.edu/)
- Tom Crane (https://orcid.org/0000-0003-1881-243X) (https://orcid.org/0000-0003-1881-243X), Digirati (http://digirati.com/)
- Robert Sanderson (https://orcid.org/0000-0003-4441-6852) (https://orcid.org/0000-0003-4441-6852), J. Paul Getty Trust (http://www.getty.edu/)
- Jon Stroop (https://orcid.org/0000-0002-0367-1243) (https://orcid.org/0000-0002-0367-1243), Princeton University Library (https://library.princeton.edu/)
- Simeon Warner (https://orcid.org/0000-0002-7970-7855) (https://orcid.org/0000-0002-7970-7855), Cornell University (https://www.cornell.edu/)

Copyright © 2012-2019 Editors and contributors. Published by the IIIF (International Image Interoperability Framework) Consortium under the CC-BY (http://creativecommons.org/licenses/by/4.0/) license, see disclaimer (/api/annex/notes/disclaimer/).

Table of Contents

1. Introduction
   1.1. Audience and Scope
   1.2. Terminology
2. URI Syntax
1. Introduction

This document describes an image delivery API (Application Programming Interface) defined by the International Image Interoperability Framework (IIIF, International Image Interoperability Framework), pronounced “Triple-Eye-Eff”) Consortium. The IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) specifies a web service that returns an image in response to a standard HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol (Secure)) request. The URI can specify the region, size, rotation, quality characteristics and format of the requested image. A URI can also be constructed to request basic technical information about the image to support client applications. This API (Application Programming Interface) was conceived of to facilitate systematic reuse of image resources in digital image repositories maintained by cultural heritage organizations. It could be adopted by any image repository or service, and can be used to retrieve static images in response to a properly constructed URI.

Please send feedback to iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com).

1.1. Audience and Scope
This document is intended for architects and developers building applications that share and consume digital images, particularly from cultural heritage institutions, museums, libraries and archives. Target applications include:

- Digital image repositories and distributed content networks.
- Image focused web applications, such as pan/zoom viewers, book-readers, etc.
- Client applications using image content for analysis or comparison.

This specification concerns image requests by a client, but not management of the images by the server. It covers how to respond to the requests given in a particular URI syntax, but does not cover methods of implementation such as rotation algorithms, transcoding, color management, compression, or how to respond to URIs that do not conform to the specified syntax. This allows flexibility for implementation in domains with particular constraints or specific community practices, while supporting interoperability in the general case.

1.2. Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, and OPTIONAL in this document are to be interpreted as described in RFC (Request for Comments) 2119 (https://www.ietf.org/rfc/rfc2119.txt).

2. URI Syntax

The IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) can be called in two ways:

- Request an image, which may be part of a larger image.
- Request information about the image, including characteristics, functionality available, and related services.

Both convey the request’s information in the path segments of the URI, rather than as query parameters. This makes responses easier to cache, either at the server or by standard web-caching infrastructure. It also permits a minimal implementation using pre-computed files in a matching directory structure.

There are four parameters shared by the requests, and other IIIF (International Image Interoperability Framework) specifications:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>Indicates the use of the HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol (Secure)) protocol in calling the service.</td>
</tr>
<tr>
<td>server</td>
<td>The host server on which the service resides. The parameter may also include a port number.</td>
</tr>
<tr>
<td>prefix</td>
<td>The path on the host server to the service. This prefix is optional, but may be useful when the host server supports multiple services. The prefix MAY contain multiple path segments, delimited by slashes, but all other special characters MUST be encoded. See URI Encoding and Decoding for more information.</td>
</tr>
<tr>
<td>identifier</td>
<td>The identifier of the requested image. This may be an ark, URN, filename, or other identifier. Special characters MUST be URI encoded.</td>
</tr>
</tbody>
</table>

The combination of these parameters forms the image’s base URI and identifies the underlying image content. It is constructed according to the following URI Template (RFC6570 (http://tools.ietf.org/html/rfc6570)):
When the base URI is dereferenced, the interaction SHOULD result in the image information document. It is RECOMMENDED that the response be a 303 status redirection to the image information document’s URI. Implementations MAY also exhibit other behavior for the base URI beyond the scope of this specification in response to HTTP (Hypertext Transfer Protocol) request headers and methods.

To allow for extensions, this specification does not define the server behavior when it receives requests that do not match either the base URI or one of the described URI syntaxes below.

2.1. Image Request URI Syntax

The IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) URI for requesting an image MUST conform to the following URI Template:

```
{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}
```

For example:

```
http://www.example.org/image-service/abcd1234/full/full/0/default.jpg
```

The parameters of the Image Request URI include region, size, rotation, quality and format, which define the characteristics of the returned image. These are described in detail in Image Request Parameters.

2.2. Image Information Request URI Syntax

The URI for requesting image information MUST conform to the following URI Template:

```
{scheme}://{server}{/prefix}/{identifier}/info.json
```

For example:

```
http://www.example.org/image-service/abcd1234/info.json
```

The scheme, server, prefix and identifier components of the information request MUST be identical to those for the image request described above for the image content that the image information document describes. The image information document is described in detail in the Image Information section.

3. Identifier

The API (Application Programming Interface) places no restrictions on the form of the identifiers that a server may use or support. All special characters (e.g. ? or #) MUST be URI encoded to avoid unpredictable client behaviors. The URI syntax relies upon slash (/) separators so any slashes in the identifier MUST be URI encoded (also called “percent encoded”). See the additional discussion in URI Encoding and Decoding.

4. Image Request Parameters

All parameters described below are required for compliant construction of a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) URI. The sequence of parameters in the URI MUST be in the order described below. The order of the parameters is also intended as a mnemonic for the order of the operations by which the service should manipulate the image content. Thus, the requested image content is first extracted as a region of the complete image, then scaled to the requested size, mirrored and/or rotated, and finally transformed into the color quality and format. This resulting image content is returned as the representation for the URI. Image and region dimensions in pixels are always given as an integer numbers. Intermediate calculations may use floating
point numbers and the rounding method is implementation specific. Some parameters, notably percentages, may be specified with floating point numbers. These should have at most 10 decimal digits and consist only of decimal digits and "." with a leading zero if less than 1.0.

4.1. Region

The region parameter defines the rectangular portion of the full image to be returned. Region can be specified by pixel coordinates, percentage or by the value “full”, which specifies that the entire image should be returned.

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>The complete image is returned, without any cropping.</td>
</tr>
<tr>
<td>square</td>
<td>The region is defined as an area where the width and height are both equal to the length of the shorter dimension of the complete image. The region may be positioned anywhere in the longer dimension of the image content at the server’s discretion, and centered is often a reasonable default.</td>
</tr>
<tr>
<td>x,y,w,h</td>
<td>The region of the full image to be returned is specified in terms of absolute pixel values. The value of x represents the number of pixels from the 0 position on the horizontal axis. The value of y represents the number of pixels from the 0 position on the vertical axis. Thus the x,y position 0,0 is the upper left-most pixel of the image. w represents the width of the region and h represents the height of the region in pixels.</td>
</tr>
<tr>
<td>pct:x,y,w,h</td>
<td>The region to be returned is specified as a sequence of percentages of the full image’s dimensions, as reported in the image information document. Thus, x represents the number of pixels from the 0 position on the horizontal axis, calculated as a percentage of the reported width. w represents the width of the region, also calculated as a percentage of the reported width. The same applies to y and h respectively. These may be floating point numbers.</td>
</tr>
</tbody>
</table>

If the request specifies a region which extends beyond the dimensions reported in the image information document, then the service SHOULD return an image cropped at the image’s edge, rather than adding empty space.

If the requested region’s height or width is zero, or if the region is entirely outside the bounds of the reported dimensions, then the server SHOULD return a 400 status code.

Examples:

1. region=full
   .../full/full/0/default.jpg

2. region=square
   .../square/full/0/default.jpg
4.2. Size

The size parameter determines the dimensions to which the extracted region is to be scaled.

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>full</strong></td>
<td>The image or region is not scaled, and is returned at its full size. Note deprecation warning.</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>The image or region is returned at the maximum size available, as indicated by <code>maxWidth</code>, <code>maxHeight</code>, <code>maxArea</code> in the profile description. This is the same as <code>full</code> if none of these properties are provided.</td>
</tr>
<tr>
<td>w, h</td>
<td>The image or region should be scaled so that its width is exactly equal to w, and the height will be a calculated value that maintains the aspect ratio of the extracted region.</td>
</tr>
<tr>
<td>pct:n</td>
<td>The width and height of the returned image is scaled to n% of the width and height of the extracted region. The aspect ratio of the returned image is the same as that of the extracted region.</td>
</tr>
<tr>
<td>w,h</td>
<td>The width and height of the returned image are exactly w and h. The aspect ratio of the returned image MAY be different than the extracted region, resulting in a distorted image.</td>
</tr>
<tr>
<td>Form</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>!w,h</td>
<td>The image content is scaled for the best fit such that the resulting width and height are less than or equal to the requested width and height. The exact scaling MAY be determined by the service provider, based on characteristics including image quality and system performance. The dimensions of the returned image content are calculated to maintain the aspect ratio of the extracted region.</td>
</tr>
</tbody>
</table>

If the resulting height or width is zero, then the server SHOULD return a 400 (bad request) status code.

The image server MAY support scaling above the full size of the extracted region.

**Deprecation Warning**  The size keyword `full` will be replaced in favor of `max` in version 3.0. Until that time, the `w`, syntax should be considered the canonical form of request for the `max` size, unless `max` is equivalent to `full`. Feedback is welcome via iiif-discuss (mailto:iiif-discuss@googlegroups.com) or on the Github issue (https://github.com/IIIF/iiif.io/issues/678).

Examples:

1. `size=full`  
   .../full/full/0/default.jpg  
   N.B. Assuming that the image has a `maxWidth` of 200px

2. `size=150,`  
   .../full/150/0/default.jpg

3. `size=150`  
   .../full/150/0/default.jpg

4. `size=pct:50`  
   .../full/pct:50/0/default.jpg

5. `size=225,100`  
   .../full/225,100/0/default.jpg

6. `size=!225,100`  
   .../full/!225,100/0/default.jpg  
   N.B. Returned image is 150,100 px
4.3. Rotation

The rotation parameter specifies mirroring and rotation. A leading exclamation mark (“!”) indicates that the image should be mirrored by reflection on the vertical axis before any rotation is applied. The numerical value represents the number of degrees of clockwise rotation, and may be any floating point number from 0 to 360.

<table>
<thead>
<tr>
<th>Form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The degrees of clockwise rotation from 0 up to 360.</td>
</tr>
<tr>
<td>!n</td>
<td>The image should be mirrored and then rotated as above.</td>
</tr>
</tbody>
</table>

A rotation value that is out of range or unsupported SHOULD result in a 400 status code.

In most cases a rotation will change the width and height dimensions of the returned image. The service SHOULD return an image that contains all of the image contents requested in the region and size parameters, even if the dimensions of the returned image file are different than specified in the size parameter. The image contents SHOULD NOT be scaled as a result of the rotation, and there SHOULD be no additional space between the corners of the rotated image contents and the bounding box of the returned image content.

For rotations which are not multiples of 90 degrees, it is RECOMMENDED that the client request the image in a format that supports transparency, such as PNG, and that the server return the image with a transparent background. There is no facility in the APL (Application Programming Interface) for the client to request a particular background color or other fill pattern.

Examples:

1. rotation=0
   
   ![Image 1](./full/full/0/default.jpg)

2. rotation=180
   
   ![Image 2](./full/full/180/default.jpg)
4.4. Quality

The quality parameter determines whether the image is delivered in color, grayscale or black and white.

<table>
<thead>
<tr>
<th>Quality</th>
<th>Parameter Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>The image is returned in full color.</td>
</tr>
<tr>
<td>gray</td>
<td>The image is returned in grayscale, where each pixel is black, white or any shade of gray in between.</td>
</tr>
<tr>
<td>bitonal</td>
<td>The image returned is bitonal, where each pixel is either black or white.</td>
</tr>
<tr>
<td>default</td>
<td>The image is returned using the server’s default quality (e.g. color, gray or bitonal) for the image.</td>
</tr>
</tbody>
</table>

The default quality exists to support level 0 compliant implementations (/api/image/2.1/compliance/#quality) that may not know the qualities of individual images in their collections. It also provides a convenience for clients that know the values for all other parameters of a
request except the quality (e.g. \(../full/120,/90/\{\text{quality}\}.png\) to request a thumbnail) in that a preliminary image information request that would only serve to find out which qualities are available can be avoided.

A quality value that is unsupported SHOULD result in a 400 status code.

Examples:

1. quality=default
   \(\ldots/full/full/0/default.jpg\)

2. quality=color
   \(\ldots/full/full/0/color.jpg\)

3. quality=gray
   \(\ldots/full/full/0/gray.jpg\)

4. quality=bitonal
   \(\ldots/full/full/0/bitonal.jpg\)

4.5. Format

The format of the returned image is expressed as an extension at the end of the URI.

<table>
<thead>
<tr>
<th>Extension</th>
<th>MIME Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpg</td>
<td>image/jpeg</td>
</tr>
<tr>
<td>tif</td>
<td>image/tiff</td>
</tr>
<tr>
<td>png</td>
<td>image/png</td>
</tr>
<tr>
<td>gif</td>
<td>image/gif</td>
</tr>
<tr>
<td>jp2</td>
<td>image/jp2</td>
</tr>
<tr>
<td>pdf</td>
<td>application/pdf</td>
</tr>
<tr>
<td>Extension</td>
<td>MIME Type</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>webp</td>
<td>image/webp</td>
</tr>
</tbody>
</table>

A format value that is unsupported SHOULD result in a 400 status code.

Examples:

1. ![full/full/0/default.jpg](.../full/full/0/default.jpg)
2. ![full/full/0/default.png](.../full/full/0/default.png)
3. ![full/full/0/default.tif](.../full/full/0/default.tif)

4.6. Order of Implementation

The sequence of parameters in the URI is intended as a mnemonic for the order in which image manipulations are made against the full image content. This is important to consider when implementing the service because applying the same parameters in a different sequence will often result in a different image being delivered. The order is critical so that the application calling the service reliably receives the output it expects.

The parameters should be interpreted as if the sequence of image manipulations were:

Region THEN Size THEN Rotation THEN Quality THEN Format

If the rotation parameter includes mirroring ("!"), the mirroring is applied before the rotation.

```
1 region=125,15,120,140 size=90, rotation=!345 quality=gray
.../125,15,120,140/90/,!345/gray.jpg
```

4.7. Canonical URI Syntax
It is possible to request the same image using different combinations of parameters. While it is useful for clients to be able to express their requests in a convenient form, there are several reasons why a canonical URI syntax is desirable:

- It enables static, file-system based implementations, which will have only a single URI at which the content is available.
- Caching becomes significantly more efficient, both client and server side, when the URIs used are the same between systems and sessions.
- Response times can be improved by avoiding redirects from a requested non-canonical URI syntax to the canonical syntax by using the canonical form directly.

In order to support the above requirements, clients SHOULD construct the image request URIs using the following canonical parameter values where possible. Image servers MAY redirect the client to the canonical URI from a non-canonical equivalent.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Canonical value</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>“full” if the whole image is requested, (including a “square” region of a square image) otherwise the (x,y,w,h) syntax.</td>
</tr>
<tr>
<td>size</td>
<td>“full” if the default size is requested, the (w), syntax for images that should be scaled maintaining the aspect ratio, and the (w,h) syntax for explicit sizes that change the aspect ratio. <strong>Note:</strong> The size keyword “full” will be replaced with “max” in version 3.0. See the deprecation warning under size for more information.</td>
</tr>
<tr>
<td>rotation</td>
<td>“!” if the image is mirrored, followed by an integer if possible, and trimming any trailing zeros in a decimal value, and a leading 0 if the value is below 1.</td>
</tr>
<tr>
<td>quality</td>
<td>“default” if the server’s default quality is requested, otherwise the quality string.</td>
</tr>
<tr>
<td>format</td>
<td>The explicit format string is always required.</td>
</tr>
</tbody>
</table>

When the client requests an image, the server MAY add a link header to the response that indicates the canonical URI for that request:

```
Link: <http://iiif.example.com/server/full/400,/0/default.jpg>;rel="canonical"
```

The server MAY include this link header on the image information response, however it is unnecessary as it is included in the JSON (JavaScript Object Notation) representation retrieved.

5. Image Information

Servers MUST support requests for image information. The response includes technical properties about the image and may also contain rights and licensing properties, and services related to it.

5.1. Image Information Request

The request for the information MUST conform to the URI Template:

```
{scheme}://{server}{/prefix}/{identifier}/info.json
```

The syntax for the response is JSON-LD (JSON for Linking Data) (http://www.w3.org/TR/json-ld/). The content-type of the response MUST be either “application/json” (regular JSON (JavaScript Object Notation)),...
If the client explicitly wants the JSON-LD (JSON for Linking Data) content-type, then it MUST specify this in an Accept header, otherwise the server MUST return the regular JSON (JavaScript Object Notation) content-type.

Servers SHOULD send the Access-Control-Allow-Origin header with the value * in response to information requests. The syntax is shown below and is described in the CORS (Cross-Origin Resource Sharing) (http://www.w3.org/TR/cors/) specification. This header is required in order to allow the JSON (JavaScript Object Notation) responses to be used by Web applications hosted on different servers.

```
Access-Control-Allow-Origin: *
```

A recipe for enabling these behaviors is provided in the Apache HTTP (Hypertext Transfer Protocol) Server Implementation Notes (/api/annex/notes/apache/#conditional-content-types).

### 5.2. Technical Properties

<table>
<thead>
<tr>
<th>Technical Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td>Required</td>
<td>The context document that describes the semantics of the terms used in the document. This must be the URI: <a href="http://iiif.io/api/image/2/context.json">http://iiif.io/api/image/2/context.json</a> for version 2.1 of the IIIF (International Image Interoperability Framework) Image API (Application Programming Interface). This document allows the response to be interpreted as RDF (Resource Description Framework), using the JSON-LD (JSON for Linking Data) serialization.</td>
</tr>
<tr>
<td>@id</td>
<td>Required</td>
<td>The base URI of the image as defined in URI Syntax, including scheme, server, prefix and identifier without a trailing slash.</td>
</tr>
<tr>
<td>@type</td>
<td>Optional</td>
<td>The type for the Image. If present, the value MUST be the string iiif:Image.</td>
</tr>
<tr>
<td>protocol</td>
<td>Required</td>
<td>The URI <a href="http://iiif.io/api/image">http://iiif.io/api/image</a> which can be used to determine that the document describes an image service which is a version of the IIIF (International Image Interoperability Framework) Image API (Application Programming Interface).</td>
</tr>
<tr>
<td>width</td>
<td>Required</td>
<td>The width in pixels of the full image content, given as an integer.</td>
</tr>
<tr>
<td>height</td>
<td>Required</td>
<td>The height in pixels of the full image content, given as an integer.</td>
</tr>
<tr>
<td>profile</td>
<td>Required</td>
<td>A list of profiles, indicated by either a URI or an object describing the features supported. The first entry in the list MUST be a compliance level URI.</td>
</tr>
<tr>
<td>sizes</td>
<td>Optional</td>
<td>A set of height and width pairs the client should use in the size parameter to request complete images at different sizes that the server has available. This may be used to let a client know the sizes that are available when the server does not support requests for arbitrary sizes, or simply as a hint that requesting an image of this size may result in a faster response. A request constructed with the w,h syntax using these sizes MUST be supported by the server, even if arbitrary width and height are not.</td>
</tr>
<tr>
<td>tiles</td>
<td>Optional</td>
<td>A set of descriptions of the parameters to use to request regions of the image (tiles) that are efficient for the server to deliver. Each description gives a width, optionally a height for non-square tiles, and a set of scale factors at which tiles of those dimensions are available.</td>
</tr>
</tbody>
</table>
The objects in the sizes list have the properties in the following table. Images requested using these sizes SHOULD have a region parameter of “full” and rotation of “0”. The size SHOULD be requested using the canonical syntax of \( w \). Thus, the full URL for an image with “default” quality in “jpg” format would be: {scheme}://{server}/{prefix}/{identifier}/full/{width},/0/default.jpg

Note that the values in \( width \) and \( height \) do not necessarily imply that an image of that size is available. If \( sizes \), \( maxArea \), \( maxWidth \), or \( maxHeight \) are present, they may indicate constraints on the maximum size of image that can be requested. The \( width \) and \( height \) information is still required in order to construct tile requests and know the aspect ratio of the image.

**Warning** There is an inconsistency between the specification of the sizes list and the canonical URI syntax. Clients SHOULD use the Canonical URI Syntax when making image requests based on entries in sizes. For maximum compatibility, servers SHOULD support both the \( w \), and \( w,h \) forms of the size parameter for values in sizes that maintain the aspect ratio. This inconsistency will be addressed in the next major version of this specification.

<table>
<thead>
<tr>
<th>Size Object Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@type</td>
<td>Optional</td>
<td>The type of the object. If present, the value MUST be the string iiif:Size.</td>
</tr>
<tr>
<td>width</td>
<td>Required</td>
<td>The width in pixels of the image to be requested, given as an integer.</td>
</tr>
<tr>
<td>height</td>
<td>Required</td>
<td>The height in pixels of the image to be requested, given as an integer.</td>
</tr>
</tbody>
</table>

The objects in the tiles list have the properties in the following table. The \( width \) and \( height \) should be used to fill the region parameter and the scaleFactors to complete the size parameter of the image URL. This is described in detail in the Implementation Notes.

The width of a tile, or the combination of \( width \) and \( height \) if \( height \) is specified, MUST be unique among the members of the tiles list.

<table>
<thead>
<tr>
<th>Tile Object Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@type</td>
<td>Optional</td>
<td>The type of the Tile. If present, the value MUST be the string iiif:Tile.</td>
</tr>
<tr>
<td>scaleFactors</td>
<td>Required</td>
<td>The set of resolution scaling factors for the image’s predefined tiles, expressed as positive integers by which to divide the full size of the image. For example, a scale factor of 4 indicates that the service can efficiently deliver images at 1/4 or 25% of the height and width of the full image. A particular scale factor value SHOULD appear only once in the tiles list.</td>
</tr>
<tr>
<td>width</td>
<td>Required</td>
<td>The width in pixels of the predefined tiles to be requested, given as an integer.</td>
</tr>
<tr>
<td>height</td>
<td>Optional</td>
<td>The height in pixels of the predefined tiles to be requested, given as an integer. If it is not specified in the JSON (JavaScript Object Notation), then it defaults to the same as ( width ), resulting in square tiles.</td>
</tr>
</tbody>
</table>

Servers SHOULD support requests for images with parameters specified by the sizes and tiles fields for all combinations of qualities and formats supported.

The following shows a valid image information response, including the optional sizes and tiles properties.
5.3. Profile Description

In order to specify additional features that are supported for the image, a profile object may be added to the profile list. Objects in the profile list have the properties in the following table. The @context, @id and @type properties are REQUIRED when the profile is dereferenced from a URI, but SHOULD NOT be included in the image information response.

<table>
<thead>
<tr>
<th>Profile Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td>Optional</td>
<td>The string “<a href="http://iiif.io/api/image/2/context.json%E2%80%9D">http://iiif.io/api/image/2/context.json”</a>. This should be included only if the profile’s URI is dereferenced.</td>
</tr>
<tr>
<td>@id</td>
<td>Optional</td>
<td>The URI of the profile.</td>
</tr>
<tr>
<td>@type</td>
<td>Optional</td>
<td>The type of the object. If present, the value MUST be the string “iiif:ImageProfile”.</td>
</tr>
<tr>
<td>formats</td>
<td>Optional</td>
<td>The set of image format parameter values available for the image. If not specified then clients should assume only formats declared in the compliance level document.</td>
</tr>
<tr>
<td>maxArea</td>
<td>Optional</td>
<td>The maximum area in pixels supported for this image. Requests for images sizes with width*height greater than this may not be supported.</td>
</tr>
<tr>
<td>maxHeight</td>
<td>Optional</td>
<td>The maximum height in pixels supported for this image. Requests for images sizes with height greater than this may not be supported. If maxWidth is specified and maxHeight is not, then clients should infer that maxHeight = maxWidth.</td>
</tr>
<tr>
<td>maxWidth</td>
<td>Optional</td>
<td>The maximum width in pixels supported for this image. Requests for images sizes with width greater than this may not be supported. MUST be specified if maxHeight is specified.</td>
</tr>
<tr>
<td>qualities</td>
<td>Optional</td>
<td>The set of image quality parameter values available for the image. If not specified then clients should assume only qualities declared in the compliance level document.</td>
</tr>
<tr>
<td>supports</td>
<td>Optional</td>
<td>The set of features supported for the image. If not specified then clients should assume only features declared in the compliance level document.</td>
</tr>
</tbody>
</table>
The `maxWidth`, `maxHeight` and `maxArea` parameters provide a way for image servers to express limits on the sizes supported for the image. If `maxWidth` alone, or `maxWidth` and `maxHeight` are specified then clients should expect requests with larger linear dimensions to be rejected. If `maxArea` is specified then clients should expect requests with larger pixel areas to be rejected. The `maxWidth / maxHeight` and `maxArea` parameters are independent, servers may implement either or both limits. Servers MUST ensure that sizes specified by any sizes or tiles properties are within any size limits expressed. Clients SHOULD NOT make requests that exceed size limits expressed.

The set of features that may be specified in the `supports` property of an Image profile are:

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseUriRedirect</td>
<td>The base URI of the service will redirect to the image information document.</td>
</tr>
<tr>
<td>canonicalLinkHeader</td>
<td>The canonical image URI HTTP (Hypertext Transfer Protocol) link header is provided on image responses.</td>
</tr>
<tr>
<td>cors</td>
<td>The CORS (Cross-Origin Resource Sharing) HTTP (Hypertext Transfer Protocol) header is provided on all responses.</td>
</tr>
<tr>
<td>jsonldMediaType</td>
<td>The JSON-LD (JSON for Linking Data) media type is provided when JSON-LD (JSON for Linking Data) is requested.</td>
</tr>
<tr>
<td>mirroring</td>
<td>The image may be rotated around the vertical axis, resulting in a left-to-right mirroring of the content.</td>
</tr>
<tr>
<td>profileLinkHeader</td>
<td>The profile HTTP (Hypertext Transfer Protocol) link header is provided on image responses.</td>
</tr>
<tr>
<td>regionByPct</td>
<td>Regions of images may be requested by percentage.</td>
</tr>
<tr>
<td>regionByPx</td>
<td>Regions of images may be requested by pixel dimensions.</td>
</tr>
<tr>
<td>regionSquare</td>
<td>A square region where the width and height are equal to the shorter dimension of the complete image content.</td>
</tr>
<tr>
<td>rotationArbitrary</td>
<td>Rotation of images may be requested by degrees other than multiples of 90.</td>
</tr>
<tr>
<td>rotationBy90s</td>
<td>Rotation of images may be requested by degrees in multiples of 90.</td>
</tr>
<tr>
<td>sizeAboveFull</td>
<td>Size of images may be requested larger than the “full” size. See warning.</td>
</tr>
<tr>
<td>sizeByConfinedWh</td>
<td>Size of images may be requested in the form “!w,h”.</td>
</tr>
<tr>
<td>sizeByDistortedWh</td>
<td>Size of images may be requested in the form “w,h”, including sizes that would distort the image.</td>
</tr>
<tr>
<td>sizeByH</td>
<td>Size of images may be requested in the form “,h”.</td>
</tr>
<tr>
<td>sizeByPct</td>
<td>Size of images may be requested in the form “pct:n”.</td>
</tr>
<tr>
<td>sizeByW</td>
<td>Size of images may be requested in the form “w,”.</td>
</tr>
<tr>
<td>sizeByWh</td>
<td>Size of images may be requested in the form “w,h” where the supplied w and h preserve the aspect ratio.</td>
</tr>
<tr>
<td>sizeByWhListed</td>
<td>See deprecation warning below.</td>
</tr>
<tr>
<td>sizeByForcedWh</td>
<td>See deprecation warning below.</td>
</tr>
</tbody>
</table>
Deprecation Warning  Use of the feature names sizeByWhListed and sizeByForcedWh is deprecated. These names will be removed in version 3.0. sizeByForcedWh was inconsistently defined in version 2.0, and sizeByWhListed is implied by listing the sizes in the image information document and is therefore not required as a named feature.

The features sizeByWh and sizeByDistortedWh share the same “w,h” syntax for the size parameter, but they represent separate features. A server that supports sizeByWh but not sizeByDistortedWh would serve an image response at any scale (subject to separate maxWidth, maxHeight, maxArea and sizeAboveFull constraints if present), but only if the resulting image preserved the original aspect ratio. Requests for distorted images would not be served.

A server that supports neither sizeByW or sizeByWh is only required to serve the image sizes listed under the sizes property or implied by the tiles property of the image information document, allowing for a static file implementation.

The set of features, formats and qualities supported is the union of those declared in all of the external profile documents and any embedded profile objects. If a feature is not present in either the profile document or the supports property of an embedded profile, then a client MUST assume that the feature is not supported.

If any of formats, qualities, or supports have no additional values beyond those specified in the referenced compliance level, then the property SHOULD be omitted from the response rather than being present with an empty list.

URIs MAY be added to the supports list of a profile to cover features not defined in this specification. Clients MUST ignore URIs that are not recognized.

The following fragment shows a profile indicating support for additional formats, qualities, and features beyond level 2 compliance. It also includes a size limit.

```json
{
    "@context" : "http://iiif.io/api/image/2/context.json",
    "@id" : "http://www.example.org/image-service/abcd1234/1E34750D-38DB-4825-A38A-B60A345E591C",
    "protocol" : "http://iiif.io/api/image",
    //...
    "profile" : [
        "http://iiif.io/api/image/2/level2.json",
        {
            "formats" : [ "gif", "pdf" ],
            "qualities" : [ "color", "gray" ],
            "maxWidth" : 2000,
            "supports" : [
                "canonicalLinkHeader", "rotationArbitrary", "profileLinkHeader",
                "http://example.com/feature/"]
        }
    ]
}
```

5.4. Rights and Licensing Properties

The rights and licensing properties, attribution, license and logo, have the same semantics and requirements as those in the Presentation API (Application Programming Interface) (/api/presentation/2.1/).
<table>
<thead>
<tr>
<th>Rights and Licensing Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribution</td>
<td>Optional</td>
<td>Text that MUST be shown when content obtained from the Image API (Application Programming Interface) service is displayed or used. It might include copyright or ownership statements, or a simple acknowledgement of the providing institution. The value MAY contain simple HTML (HyperText Markup Language) as described in the HTML (HyperText Markup Language) Markup in Property Values (api/presentation/2.1/#html-markup-in-property-values) section of the Presentation API (Application Programming Interface).</td>
</tr>
<tr>
<td>license</td>
<td>Optional</td>
<td>A link to an external resource that describes the license or rights statement under which content obtained from the Image API (Application Programming Interface) service may be used.</td>
</tr>
<tr>
<td>logo</td>
<td>Optional</td>
<td>A small image that represents an individual or organization associated with the content. Logo images MUST be clearly rendered when content obtained from the Image API (Application Programming Interface) service is displayed or used. Clients MUST NOT crop, rotate, or otherwise distort the image.</td>
</tr>
</tbody>
</table>

All of the rights and licensing properties MAY have multiple values, expressed as a JSON (JavaScript Object Notation) array, or a single value.

In the case where multiple values are supplied for attribution, clients MUST use the following algorithm to determine which values to display to the user:

- If none of the values have a language associated with them, the client MUST display all of the values.
- Else, the client should try to determine the user’s language preferences, or failing that use some default language preferences. Then:
  - If any of the values have a language associated with them, the client MUST display all of the values associated with the language that best matches the language preference.
  - If all of the values have a language associated with them, and none match the language preference, the client MUST select a language and display all of the values associated with that language.
  - If some of the values have a language associated with them, but none match the language preference, the client MUST display all of the values that do not have a language associated with them.

The value of the logo property may be a string containing the URL of the image, or a JSON (JavaScript Object Notation) object that indicates the URI of both the logo image and a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) service for the logo. While possible, it is RECOMMENDED that logos with IIIF (International Image Interoperability Framework) services do not, themselves, have logos. Clients encountering logos with logos are not required to display a potentially infinite set.

When both the Image and Presentation APIs express attributions or logos, then clients MUST display both unless they are identical.

The following shows a simple use of each of these properties:
More complex examples are given in the **Complete Response Example**.

### 5.5. Related Services

<table>
<thead>
<tr>
<th>Property</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Optional</td>
<td>The service property provides hooks for additional information to be included in the image description, for example a link to an authentication service. The value may be an object or a list of objects.</td>
</tr>
</tbody>
</table>

There MAY be one or more services associated with an image. See the Service Profiles (/api/annex/services/) annex for more information.

The following shows a use of `service` to associate the login page of an authentication system that users must go through in order to access the image. For further information, please see **Authentication**.

```json
{
    "@context": "http://iiif.io/api/image/2/context.json",
    "@id": "http://www.example.org/image-service/abcd1234/1E34750D-38DB-4825-A38A-B60A345E591C",
    "protocol": "http://iiif.io/api/image",
    "service": {
        "@context": "http://iiif.io/api/auth/1/context.json",
        "@id": "http://www.example.org/auth/login.html",
        "profile": "http://iiif.io/api/auth/1/login"
    }
}
```

More complex examples are given in the **Complete Response Example**.

### 5.6. Complete Response

The following shows a response including all of the required and optional image information properties.
6. Compliance Levels

The image information document MUST specify the extent to which the API (Application Programming Interface) is supported by including a compliance level URI as the first entry in the profile property. This URI links to a description of the highest compliance level for which all requirements are met. The URI MUST be one of those listed in the Image API (Application Programming Interface) Compliance (/api/image/2.1/compliance/) document. This description contains the set of features required by the profile, as discussed in the Image Information section. A server MAY declare different compliance levels for images with different identifiers.

The compliance level URI MAY also be given in the HTTP (Hypertext Transfer Protocol) Link header (RFC5988 (http://tools.ietf.org/html/rfc5988)) with the parameter rel="profile", and thus a complete header might look like:

```
Link: <http://iiif.io/api/image/2/level1.json>;rel="profile"
```

A recipe for setting this header on the Apache HTTP (Hypertext Transfer Protocol) Server is shown in the Apache HTTP (Hypertext Transfer Protocol) Server Implementation Notes (/api/annex/notes/apache/#set-compliance-link-header).

7. Server Responses

7.1. Successful Responses

Servers may transmit HTTP (Hypertext Transfer Protocol) responses with 200 (Successful) or 3xx (Redirect) status codes when the request has been successfully processed. If the status code is 200, then the entity-body MUST be the requested image or information document. If the status code is 301, 302, 303, or 304, then the entity-body is unrestricted, but it is RECOMMENDED to be empty. If the status code is 301, 302, or 303 then the Location HTTP (Hypertext Transfer Protocol) Header MUST be set containing the URI of the image that fulfills the request. This enables servers to have a single canonical URI to promote caching of responses. Status code 304 is handled exactly as per the HTTP (Hypertext Transfer Protocol) specification. Clients SHOULD expect to encounter all of these situations and MUST NOT assume that the entity-body of the initial response necessarily contains the image data.

7.2. Error Conditions

The order in which servers parse requests and detect errors is not specified. A request is likely to fail on the first error encountered and return an appropriate HTTP (Hypertext Transfer Protocol) status code, with common codes given in the list below. It is RECOMMENDED that the body of the error response includes a human-readable description of the error in either plain text or html.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400 Bad Request</td>
<td>The server cannot fulfill the request, as the syntax of the request issued by the client is incorrect.</td>
</tr>
<tr>
<td>401 Unauthorized</td>
<td>Authentication is required and not provided. See the Authentication section for details.</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>The user, authenticated or not, is not permitted to perform the requested operation.</td>
</tr>
</tbody>
</table>
### Status Codes

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>404 Not Found</td>
<td>The image resource specified by identifier does not exist, the value of one or more of the parameters is not supported for this image, or the requested size is greater than the limits specified.</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>The server encountered an unexpected error that prevented it from fulfilling the request.</td>
</tr>
<tr>
<td>501 Not Implemented</td>
<td>The server received a valid IIIF (International Image Interoperability Framework) request that is not implemented.</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>The server is busy/temporarily unavailable due to load/maintenance issues.</td>
</tr>
</tbody>
</table>

### 8. Authentication

Images are generally secondary resources in a web page or application. In the case of web pages, images are embedded in the HTML (HyperText Markup Language) `<img>` tag, and are retrieved via additional HTTP (Hypertext Transfer Protocol) requests. When a user cannot load a web page, it is possible — and a generally accepted behavior — to redirect the user to another page and offer the opportunity to authenticate. This is not an option for secondary resources such as images, and the user is instead simply presented with a broken image icon.

No new authentication mechanisms are proposed, nor roles for authorization business logic. Instead, it is expected that authentication requirements and processes are handled outside of any IIIF (International Image Interoperability Framework)-specific context, but within a IIIF (International Image Interoperability Framework)-aware access control workflow. Please see the draft authentication (/api/auth/) specification.

### 9. URI Encoding and Decoding

The URI syntax of this API (Application Programming Interface) relies upon slash (/) separators which MUST NOT be encoded. Clients MUST percent-encode special characters (the to-encode set below: percent and gen-delims of RFC3986 (http://tools.ietf.org/html/rfc3986) except the colon) plus any characters outside the US-ASCII set within the components of requests. For example, any slashes within the identifier part of the URI MUST be percent-encoded. Encoding is necessary only for the identifier because other components will not include special characters. Percent-encoding other characters introduces no ambiguity but is unnecessary.

```plaintext
to-encode = "/" / "?" / "#" / "[" / "]" / "@" / "%"
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>URI path</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>identifier=id1 region=full size=full rotation=0 quality=default</code></td>
<td>id1/full/full/0/default</td>
</tr>
<tr>
<td><code>identifier=id1 region=0,10,100,200 size=pct:50 rotation=90 quality=default format=png</code></td>
<td>id1/0,10,100,200/pct:50/90/default.png</td>
</tr>
<tr>
<td><code>identifier=id1 region=pct:10,10,80,80 size=50, rotation=22.5 quality=color format=jpg</code></td>
<td>id1/pct:10,10,80,80/50,/22.5/color.jpg</td>
</tr>
<tr>
<td><code>identifier=bb157hs6068 region=full size=full rotation=270 quality=gray format=jpg</code></td>
<td>bb157hs6068/full/full/270/gray.jpg</td>
</tr>
</tbody>
</table>
Servers which are incapable of processing arbitrarily encoded identifiers SHOULD make their best efforts to expose only image identifiers for which clients will not encode any of the characters, and thus it is RECOMMENDED to limit characters in identifiers to letters, numbers and the underscore character.

10. Security Considerations

This API (Application Programming Interface) defines a URI syntax and the semantics associated with its components. The composition of URIs has few security considerations except possible exposure of sensitive information in URIs or revealing of browse/view behavior of users.

Server applications implementing this API (Application Programming Interface) should consider possible denial-of-service attacks, and authentication vulnerabilities based on DNS spoofing. Applications must be careful to parse and sanitize incoming requests (URIs) in ways that avoid overflow, injection, and directory traversal attacks.

It is recommended that servers implementing the sizeAboveFull feature also implement one or more of maxWith, maxHeight, or maxArea in order to prevent arbitrarily large image requests, thus exposing the server to denial-of-service attacks.

Early sanity checking of URIs (lengths, trailing GET, invalid characters, out-of-range parameters) and rejection with appropriate response codes is recommended.

11. Appendices

A. Implementation Notes

- For use cases that enable the saving of the image, it is RECOMMENDED to use the HTTP (Hypertext Transfer Protocol) Content-Disposition header (RFC6266 (http://tools.ietf.org/html/rfc6266)) to provide a convenient filename that distinguishes the image, based on the identifier and parameters provided.

- Server implementations may rely on components or frameworks that unescape the URI path, such as Python's WSGI (https://www.python.org/dev/peps/pep-0333/). In such situations, the requested URI may be parsed from the right in order to handle identifiers possibly containing slashes, given the knowledge of the API (Application Programming Interface) parameters and the prefix for which the server handles requests.

- This specification makes no assertion about the rights status of requested images or any other descriptive metadata, whether or not authentication has been accomplished. Please see the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) (/api/presentation/2.1/) for rights and other information.
- Additional Apache HTTP (Hypertext Transfer Protocol) Server implementation notes (/api/annex/notes/apache/) are available.
- Linked data implementations may construct the info.json response using the frame supplied in the JSON-LD (JSON for Linking Data) framing implementation note (/api/annex/notes/jsonld/).
- When requesting sizes using the w, canonical syntax, if a particular height is desired, the following algorithm can be used:

  ```python
  # Calculate request width for `w`, syntax from desired height
  request_width = image_width * desired_height / image_height
  ```

- When requesting image tiles, the Region and Size parameters must be calculated to take account of partial tiles along the right and lower edges for a full image that is not an exact multiple of the scaled tile size. The algorithm below is shown as Python code and assumes integer inputs and integer arithmetic throughout (ie. remainder discarded on division). Inputs are: size of full image content (width,height), scale factor s, tile size (tw,th), and tile coordinate (n,m) counting from (0,0) in the upper-left corner. Note that the rounding method is implementation dependent.

  ```python
  # Calculate region parameters /xr,yr,wr,hr/
  xr = n * tw * s
  yr = m * th * s
  wr = tw * s
  if (xr + wr > width):
    wr = width - xr
  hr = th * s
  if (yr + hr > height):
    hr = height - yr
  # Calculate size parameters /ws,hs/
  ws = tw
  if (xr + tw*s > width):
    ws = (width - xr + s - 1) / s  # +s-1 in numerator to round up
  hs = th
  if (yr + th*s > height):
    hs = (height - yr + s - 1) / s
  ```

- As described in Rotation, in order to retain the size of the requested image contents, rotation will change the width and height dimensions of the image returned. A formula for calculating the dimensions of the image returned for a given starting size and rotation is given below. Note that the rounding method is implementation dependent and that some languages require conversion of the angle from degrees to radians.

  ```python
  # (w,h) are size parameters, n is rotation angle
  w_returned = abs(w*cos(n)) + abs(h*sin(n))
  h_returned = abs(h*cos(n)) + abs(w*sin(n))
  ```

B. Versioning

Starting with version 2.0, this specification follows Semantic Versioning (http://semver.org/spec/v2.0.0.html). See the note Versioning of APIs (/api/annex/notes/semver/) for details regarding how this is implemented.

C. Acknowledgments

The production of this document was generously supported by a grant from the Andrew W. Mellon Foundation (http://www.mellon.org/).
Many thanks to the members of the IIIF (International Image Interoperability Framework) (https://iiif.io/community/) for their continuous engagement, innovative ideas and feedback.

D. Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-06-09</td>
<td>Version 2.1.1 View change log (/api/image/2.1/change-log-211/)</td>
</tr>
<tr>
<td>2016-05-12</td>
<td>Version 2.1 (Crowned Eagle) View change log (/api/image/2.1/change-log/)</td>
</tr>
<tr>
<td>2014-09-11</td>
<td>Version 2.0 (Voodoo Bunny) View change log (/api/image/2.0/change-log/)</td>
</tr>
<tr>
<td>2013-09-17</td>
<td>Version 1.1 (unnamed) View change log (/api/image/1.1/change-log/)</td>
</tr>
<tr>
<td>2012-08-10</td>
<td>Version 1.0 (unnamed)</td>
</tr>
</tbody>
</table>

Feedback: iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com) Get involved: Join IIIF (https://iiif.io/community/#how-to-get-involved)
IIIF Presentation API 2.1.1

Status of this Document

This Version: 2.1.1
Latest Stable Version: 2.1.1 (/api/presentation/2.1/)
Previous Version: 2.0 (/api/presentation/2.0/)

Editors

Michael Appleby (https://orcid.org/0000-0002-1266-298X), Yale University (http://www.yale.edu/)
Tom Crane (https://orcid.org/0000-0003-1881-243X), Digirati (http://digirati.com/)
Robert Sanderson (https://orcid.org/0000-0003-4441-6852), J. Paul Getty Trust (http://www.getty.edu/)
Jon Stroop (https://orcid.org/0000-0002-0367-1243), Princeton University Library (https://library.princeton.edu/)
Simeon Warner (https://orcid.org/0000-0002-7970-7855), Cornell University (https://www.cornell.edu/)

Copyright © 2012-2019 Editors and contributors. Published by the IIIF (International Image Interoperability Framework) Consortium under the CC-BY (http://creativecommons.org/licenses/by/4.0/) license, see disclaimer (/api/annex/notes/disclaimer/).

Table of Contents

1. Introduction
   1.1. Objectives and Scope
   1.2. Motivating Use Cases
   1.3. Terminology
2. Resource Type Overview
   2.1. Basic Types
   2.2. Additional Types
3. Resource Properties
   3.1. Descriptive Properties
   3.2. Rights and Licensing Properties
   3.3. Technical Properties
   3.4. Linking Properties
   3.5. Paging Properties
4. JSON-LD (JSON for Linking Data) Considerations
   4.1. URI Representation
   4.2. Repeated Properties
   4.3. Language of Property Values
   4.4. HTML (HyperText Markup Language) Markup in Property Values
   4.5. Linked Data Context and Extensions
5. Resource Structure
1. Introduction

Access to image-based resources is fundamental to many research disciplines, scholarship and the transmission of cultural knowledge. Digital images are a container for much of the information content in the Web-based delivery of museum objects, books, newspapers, letters, manuscripts, maps, scrolls, single sheet collections, and digital surrogates of textiles, realia and ephemera. Collections of born-digital images can also benefit from a standardized method to structure their layout and presentation, such as slideshows, image carousels, web comics, and more.

This document describes how the structure and layout of a complex image-based object can be made available in a standard manner. Many different styles of viewer can be implemented that consume the information to enable a rich and dynamic experience, consuming content from across collections and hosting institutions.

An object may comprise a series of pages, surfaces or other views; for example the single view of a painting, the two sides of a photograph, four cardinal views of a statue, or the many pages of an edition of a newspaper or book. The primary requirements for the Presentation API (Application Programming Interface) are to provide an order for these views, the resources needed to display a representation of the view, and the descriptive information needed to allow the user to understand what is being seen.

The principles of Linked Data (http://linkeddata.org/) and the Architecture of the Web (http://www.w3.org/TR/webarch/) are adopted in order to provide a distributed and interoperable system. The Shared Canvas data model (/model/shared-canvas/1.0) and JSON-LD (JSON for Linking Data) (http://www.w3.org/TR/json-ld/) are leveraged to create an easy-to-implement, JSON (JavaScript Object Notation)-based format.

Please send feedback to iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com)

1.1. Objectives and Scope

The objective of the IIIF (International Image Interoperability Framework) (pronounced “Triple-Eye-Eff”) Presentation API (Application Programming Interface) is to provide the information necessary to allow a rich, online viewing environment for primarily image-based objects to be presented to a human user, likely in conjunction with the IIIF Image API (Application Programming Interface) (/api/image/2.1/). This is the sole purpose of the API (Application Programming Interface) and therefore the descriptive information is given in a way that is intended for humans to read, but not semantically available to machines. In particular, it explicitly does not aim to provide metadata that would drive discovery of the digitized objects.

The following are within the scope of the current document:

- The display of digitized images associated with a particular physical object, or born-digital compound object.
- Navigation between the pages, surfaces or views of the object.
- The display of text, and resources of other media types, associated with the object or its pages – this includes descriptive information about the object, labels that can aid navigation such as numbers associated with individual pages, copyright or attribution information,
The following are not within scope:

- The discovery or selection of interesting digitized objects is not directly supported; however hooks to reference further resources are available.
- Search within the object is described by the IIIF (International Image Interoperability Framework) Content Search API (Application Programming Interface) (/api/search/1.0/).

Note that in the following descriptions, “object” (or “physical object”) is used to refer to a physical object that has been digitized or a born-digital compound object, and “resources” refer to the digital resources that are the result of that digitization or digital creation process.

1.2. Motivating Use Cases

There are many different types of digitized or digital compound objects, from ancient scrolls to modern newspapers, from medieval manuscripts to online comics, and from large maps to small photographs. Many of them bear texts, sometimes difficult to read either due to the decay of the physical object or lack of understanding of the script or language. These use cases are described in a separate document (/api/presentation/usecases/).

Collectively the use cases require a model in which one can characterize the object (via the manifest resource), the order in which individual surfaces or views are presented (the sequence resource), and the individual surfaces or views (canvas resources). Each canvas may have images and/or other content resources associated with it (content resources) to allow the view to be rendered. An object may also have parts; for example, a book may have chapters where several pages may be associated with a single chapter (a range resource) or there may be groups of content resource above the page level, such as all of the texts that make up a single edition of a book (a layer resource). These resource types, along with their properties, make up the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface).

1.3. Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC (Request for Comments) 2119 (http://tools.ietf.org/html/rfc2119).

2. Resource Type Overview

This section provides an overview of the resource types (or classes) that are used in the specification. They are each presented in more detail in Section 5.

2.1. Basic Types

This specification makes use of the following primary resource types:

**Manifest**
The overall description of the structure and properties of the digital representation of an object. It carries information needed for the viewer to present the digitized content to the user, such as a title and other descriptive information about the object or the intellectual work that it conveys. Each manifest describes how to present a single object such as a book, a photograph, or a statue.

**Sequence**
The order of the views of the object. Multiple sequences are allowed to cover situations when there are multiple equally valid orders through the content, such as when a manuscript’s pages are rebound or archival collections are reordered.

**Canvas**
A virtual container that represents a page or view and has content resources associated with it or with parts of it. The canvas provides a frame of reference for the layout of the content. The concept of a canvas is borrowed from standards like PDF and HTML (HyperText Markup Language), or applications like Photoshop and Powerpoint, where the display starts from a blank canvas and images, text and other resources are “painted” on to it.

**Content**
Content resources such as images or texts that are associated with a canvas.

2.2. Additional Types

**Collection**
An ordered list of manifests, and/or further collections. Collections allow easy advertising and browsing of the manifests in a hierarchical structure, potentially with its own descriptive information. They can also provide clients with a means to locate all of the manifests known to the publishing institution.

**Annotation**
Content resources and commentary are associated with a canvas via an annotation. This provides a single, coherent method for aligning information, and provides a standards based framework for distinguishing parts of resources and parts of canvases. As annotations can be added later, it promotes a distributed system in which publishers can align their content with the descriptions created by others.

**AnnotationList**
An ordered list of annotations, typically associated with a single canvas.

**Layer**
An ordered list of annotation lists. Layers allow higher level groupings of annotations to be recorded. For example, all of the English translation annotations of a medieval French document could be kept separate from the transcription or an edition in modern French.

**Range**
An ordered list of canvases, and/or further ranges. Ranges allow canvases, or parts thereof, to be grouped together in some way. This could be for textual reasons, such as to distinguish books, chapters, verses, sections, non-content-bearing pages, the table of contents or similar. Equally, physical features might be important such as quires or gatherings, sections that have been added later and so forth.

3. Resource Properties

This specification defines properties in five distinct areas. Most of the properties may be associated with any of the resource types described above, and may have more than one value. The property relates to the resource that it is associated with, so a description property on a manifest is a description of the object, whereas a description property on a canvas is a description of that particular page or view of the object.

The requirements for the use of the properties are summarized in Appendix B.

Other properties are allowed, either via custom extensions or endorsed by the IIIF (International Image Interoperability Framework). If a client discovers properties that it does not understand, then it MUST ignore them. Other properties SHOULD consist of a prefix and a name in the form "prefix:name" to ensure it does not collide with a property defined by IIIF (International Image Interoperability Framework) specifications. Services (/api/annex/services/) SHOULD be used for extensions if at all possible, and a JSON-LD (JSON for Linking Data) context document should be added that defines the semantics of the new properties.

3.1. Descriptive Properties

label

A human readable label, name or title for the resource. This property is intended to be displayed as a short, textual surrogate for the resource if a human needs to make a distinction between it and similar resources, for example between pages or between a choice of images to display.

- A collection MUST have at least one label.
- A manifest MUST have at least one label, such as the name of the object or title of the intellectual work that it embodies.
- A sequence MAY have one or more labels, and if there are multiple sequences in a single manifest then they MUST each have at least one label.
- A canvas MUST have at least one label, such as the page number or short description of the view.
- A content resource MAY have one or more labels, and if there is a choice of content resource for the same canvas, then they SHOULD each have at least one label.
- A layer MUST have at least one label.
- Other resource types MAY have labels.

metadata

A list of short descriptive entries, given as pairs of human readable label and value to be displayed to the user. The value SHOULD be either simple HTML (HyperText Markup Language), including links and text markup, or plain text. There are no semantics conveyed by this information, and clients SHOULD NOT use it for discovery or other purposes. This list of descriptive pairs SHOULD be able to be displayed in a tabular form in the user interface. Clients SHOULD have a way to display the information about manifests and canvases, and MAY have a way to view the information about other resources. The client SHOULD display the pairs in the order provided by the description. A pair might be used to convey the author of the work, information about its creation, a brief physical description, or ownership information, amongst other use cases. The client is not expected to take any action on this information beyond displaying the label and value. An example pair of label and value might be a label of “Author” and a value of “Jehan Froissart”.

- A collection SHOULD have one or more metadata pairs associated with it.
- A manifest SHOULD have one or more metadata pairs associated with it describing the object or work.
- Other resource types MAY have one or more metadata pairs.

description

A longer-form prose description of the object or resource that the property is attached to, intended to be conveyed to the user as a full text description, rather than a simple label and value. It MAY be in simple HTML (HyperText Markup Language) or plain text. It can duplicate any of the information from the metadata fields, along with additional information required to understand what is being displayed. Clients SHOULD have a way to display the descriptions of manifests and canvases, and MAY have a way to view the information about other resources.

- A collection SHOULD have one or more descriptions.
- A manifest SHOULD have one or more descriptions.
- Other resource types MAY have one or more description.

thumbnail

A small image that depicts or pictorially represents the resource that the property is attached to, such as the title page, a significant image or rendering of a canvas with multiple content resources associated with it. It is RECOMMENDED that a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service be available for this image for manipulations such as...

* Also available via "members" N.B. These properties will be deprecated in 3.0
resizing. If a resource has multiple thumbnails, then each of them SHOULD be different.

- A collection SHOULD have exactly one thumbnail image, and MAY have more than one.
- A manifest SHOULD have exactly one thumbnail image, and MAY have more than one.
- A sequence MAY have one or more thumbnails and SHOULD have at least one thumbnail if there are multiple sequences in a single manifest.
- A canvas MAY have one or more thumbnails and SHOULD have at least one thumbnail if there are multiple images or resources that make up the representation.
- A content resource MAY have one or more thumbnails and SHOULD have at least one thumbnail if it is an option in a choice of resources.
- Other resource types MAY have one or more thumbnails.

3.2. Rights and Licensing Properties

The following properties ensure that the interests of the owning or publishing institutions are conveyed regardless of the viewing environment, and a client MUST make these properties clearly available to the user. Given the wide variation of potential client user interfaces, it will not always be possible to display all or any of the properties to the user in the client’s initial state. If initially hidden, the method of revealing them MUST be obvious, such as a button or scroll bars.

**attribution**

Text that MUST be shown when the resource it is associated with is displayed or used. For example, this could be used to present copyright or ownership statements, or simply an acknowledgement of the owning and/or publishing institution. Clients SHOULD try to match the language preferred by the user, and if the preferred language is unknown or unavailable, then the client may choose which value to display. If there are multiple values of the same or unspecified language, then all of those values MUST be displayed.

- Any resource type MAY have one or more attribution labels.

**license**

A link to an external resource that describes the license or rights statement under which the resource may be used. The rationale for this being a URI and not a human readable label is that typically there is one license for many resources, and the text is too long to be displayed to the user along with the object. If displaying the text is a requirement, then it is RECOMMENDED to include the information using the attribution property instead.

- Any resource type MAY have one or more licenses associated with it.

**logo**

A small image that represents an individual or organization associated with the resource it is attached to. This could be the logo of the owning or hosting institution. The logo MUST be clearly rendered when the resource is displayed or used, without cropping, rotating or otherwise distorting the image. It is RECOMMENDED that a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) API (Application Programming Interface) (/api/image/2.1/) service be available for this image for manipulations such as resizing.

- Any resource type MAY have one or more logos associated with it.

3.3. Technical Properties

**@id**

The URI that identifies the resource. It is RECOMMENDED that an HTTP (Hypertext Transfer Protocol) URI be used for all resources. Recommended HTTP (Hypertext Transfer Protocol) URI patterns for the different classes of resource are given below. URIs from any registered scheme (http://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml) MAY be used, and implementers may find it convenient to use a UUID URN (http://tools.ietf.org/html/rfc4122) of the form: "urn:uuid:uuid-goes-here-1234". Resources that do not require URIs MAY be assigned blank node identifiers (http://www.w3.org/TR/rdf11-concepts/#section-blank-nodes); this is the same as omitting @id.

- A collection MUST have exactly one id, and it MUST be the http(s) URI at which it is published.
- A manifest MUST have exactly one id, and it MUST be the http(s) URI at which it is published.
- A sequence MAY have an id and MUST NOT have more than one.
- A canvas MUST have exactly one id, and it MUST be an http(s) URI. The canvas’s JSON (JavaScript Object Notation) representation SHOULD be published at that URI.
- A content resource MUST have exactly one id unless it is embedded in the response, and it MUST be the http(s) URI at which the resource is published.
- A range MUST have exactly one id, and it MUST be an http(s) URI.
- A layer MUST have exactly one id, and it MUST be an http(s) URI.
- An annotation list MUST have exactly one id, and it MUST be the http(s) URI at which it is published.
- An annotation SHOULD have exactly one id, MUST NOT have more than one, and the annotation’s representation SHOULD be published at that URI.

**@type**

The type of the resource. For the resource types defined by this specification, the value of @type will be described in the sections below. For content resources, the type may be drawn from other vocabularies. Recommendations for basic types such as image, text or audio are also given in the sections below.

- All resource types MUST have at least one type specified.
This requirement applies only to the types described in Section 2. Services, Thumbnails and other resources will have their own requirements.

format
The specific media type (often called a MIME type) of a content resource, for example “image/jpeg”. This is important for distinguishing text in XML (eXtensible Markup Language) from plain text, for example.

- A content resource MAY have exactly one format, and if so, it MUST be the value of the Content-Type header returned when the resource is dereferenced.
- Other resource types MUST NOT have a format.

This is different to the formats property in the Image API (Application Programming Interface) (/api/image/2.1), which gives the extension to use within that API (Application Programming Interface). It would be inappropriate to use in this case, as format can be used with any content resource, not just images.

height
The height of a canvas or image resource. For images, the value is in pixels. For canvases, the value does not have a unit. In combination with the width, it conveys an aspect ratio for the space in which content resources are located.

- A canvas MUST have exactly one height.
- Content resources MAY have exactly one height, given in pixels, if appropriate.
- Other resource types MUST NOT have a height.

width
The width of a canvas or image resource. For images, the value is in pixels. For canvases, the value does not have a unit. In combination with the height, it conveys an aspect ratio for the space in which content resources are located.

- A canvas MUST have exactly one width.
- Content resources MAY have exactly one width, given in pixels, if appropriate.
- Other resource types MUST NOT have a width.

viewingDirection
The direction that a sequence of canvases SHOULD be displayed to the user. Possible values are specified in the table below.

- A manifest MAY have exactly one viewing direction, and if so, it applies to all of its sequences unless the sequence specifies its own viewing direction.
- A sequence MAY have exactly one viewing direction.
- A range or layer MAY have exactly one viewing direction.
- Other resource types MUST NOT have a viewing direction.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left-to-right</td>
<td>The object is displayed from left to right.</td>
</tr>
<tr>
<td>right-to-left</td>
<td>The object is displayed from right to left.</td>
</tr>
<tr>
<td>top-to-bottom</td>
<td>The object is displayed from the top to the bottom.</td>
</tr>
<tr>
<td>bottom-to-top</td>
<td>The object is displayed from the bottom to the top.</td>
</tr>
</tbody>
</table>

viewingHint
A hint to the client as to the most appropriate method of displaying the resource. This specification defines the values specified in the table below. Other values MAY be given, and if they are, they MUST be URLs.

- Any resource type MAY have one or more viewing hints.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>individuals</td>
<td>Valid on collection, manifest, sequence and range. When used as the viewingHint of a collection, the client should treat each of the manifests as distinct individual objects. For manifest, sequence and range, the canvases referenced are all distinct individual views, and SHOULD NOT be presented in a page-turning interface. Examples include a gallery of paintings, a set of views of a 3 dimensional object, or a set of the front sides of photographs in a collection.</td>
</tr>
<tr>
<td>paged</td>
<td>Valid on manifest, sequence and range. Canvases with this viewingHint represent pages in a bound volume, and SHOULD be presented in a page-turning interface if one is available. The first canvas is a single view (the first recto) and thus the second canvas represents the back of the object in the first canvas.</td>
</tr>
<tr>
<td>continuous</td>
<td>Valid on manifest, sequence and range. A canvas with this viewingHint is a partial view and an appropriate rendering might display either the canvases individually, or all of the canvases virtually stitched together in the display. Examples when this would be appropriate include long scrolls, rolls, or objects designed to be displayed adjacent to each other. If this viewingHint is present, then the resource MUST also have a viewingDirection which will determine the arrangement of the canvases. Note that this does not allow for both sides of a scroll to be included in the same manifest with this viewingHint. To accomplish that, the manifest should be &quot;individuals&quot; and have two ranges, one for each side, which are &quot;continuous&quot;.</td>
</tr>
<tr>
<td>multi-part</td>
<td>Valid only for collections. Collections with this viewingHint consist of multiple manifests that each form part of a logical whole. Clients might render the collection as a table of contents, rather than with thumbnails. Examples include multi-volume books or a set of journal issues or other serials.</td>
</tr>
<tr>
<td>non-paged</td>
<td>Valid only for canvases. Canvases with this viewingHint MUST NOT be presented in a page turning interface, and MUST be skipped over when determining the page sequence. This viewing hint MUST be ignored if the current sequence or manifest does not have the &quot;paged&quot; viewing hint.</td>
</tr>
<tr>
<td>top</td>
<td>Valid only for ranges. A Range with this viewingHint is the top-most node in a hierarchy of ranges that represents a structure to be rendered by the client to assist in navigation. For example, a table of contents within a paged object, major sections of a 3d object, the textual areas within a single scroll, and so forth. Other ranges that are descendants of the &quot;top&quot; range are the entries to be rendered in the navigation structure. There MAY be multiple ranges marked with this hint. If so, the client SHOULD choose a style of multiple structures to navigate through.</td>
</tr>
<tr>
<td>facing-pages</td>
<td>Valid only for canvases. Canvases with this viewingHint, in a sequence or manifest with the &quot;paged&quot; viewing hint, MUST be displayed by themselves, as they depict both parts of the opening. If all of the canvases are like this, then page turning is not possible, so simply use &quot;individuals&quot; instead.</td>
</tr>
</tbody>
</table>

**navDate**

A date that the client can use for navigation purposes when presenting the resource to the user in a time-based user interface, such as a calendar or timeline. The value MUST be an `xsd:dateTime` literal in UTC, expressed in the form “YYYY-MM-DDThh:mm:ssZ”. If the exact time is not known, then “00:00:00” SHOULD be used. Similarly, the month or day SHOULD be 01 if not known. There MUST be at most one navDate associated with any given resource. More descriptive date ranges, intended for display directly to the user, SHOULD be included in the metadata property for human consumption.  
- A collection or manifest MAY have exactly one navigation date associated with it.  
- Other resource types MUST NOT have navigation dates.

### 3.4. Linking Properties

**related**

A link to an external resource intended to be displayed directly to the user, and is related to the resource that has the related property. Examples might include a video or academic paper about the resource, a website, an HTML (HyperText Markup Language) description, and so forth. A label and the format of the related resource SHOULD be given to assist clients in rendering the resource to the user.  
- Any resource type MAY have one or more external resources related to it.

**rendering**

A link to an external resource intended for display or download by a human user. This property can be used to link from a manifest, collection or other resource to the preferred viewing environment for that resource, such as a viewer page on the publisher’s web site. Other uses include a rendering of a manifest as a PDF or EPUB with the images and text of the book, or a slide deck with images of the museum object. A label and the format of the rendering resource MUST be supplied to allow clients to present the option to the user.  
- Any resource type MAY have one or more external rendering resources.

**service**

A link to a service that makes more functionality available for the resource, such as from an image to the base URI of an associated IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service. The service resource SHOULD have additional information associated with it in order to allow the client to determine how to make appropriate use of it, such as a profile link to a service description. It MAY also have relevant information copied from the service itself. This duplication is permitted in order to increase the performance of rendering the object without necessitating additional HTTP (Hypertext Transfer Protocol) requests. Please see the Service Profiles (/api/annex/services/) document for known services.  
- Any resource type MAY have one or more links to an external service.

**seeAlso**
A link to a machine readable document that semantically describes the resource with the seeAlso property, such as an XML (eXtensible Markup Language) or RDF (Resource Description Framework) description. This document could be used for search and discovery or inferencing purposes, or just to provide a longer description of the resource. The profile and format properties of the document SHOULD be given to help the client to make appropriate use of the document.

- Any resource type MAY have one or more external descriptions related to it.

**within**
A link to a resource that contains the current resource, such as annotation lists within a layer. This also allows linking upwards to collections that allow browsing of the digitized objects available.

- Collections or annotation lists that serve as pages MUST be within exactly one paged resource.
- Other resource types, including collections or annotation lists not serving as pages, MAY be within one or more containing resources.

**startCanvas**
A link from a sequence or range to a canvas that is contained within the sequence. On seeing this relationship, a client SHOULD advance to the specified canvas when beginning navigation through the sequence/range. This allows the client to begin with the first canvas that contains interesting content rather than requiring the user to skip past blank or empty canvases manually.

- A sequence or a range MAY have exactly one canvas as its start canvas.
- Other resource types MUST NOT have a start canvas.

**contentLayer**
A link from a range to a layer that includes the annotations of content resources for that range. Clients might use this to present content to the user from a different canvas when interacting with the range, or to jump to the next part of the range within the same canvas.

- A range MAY have exactly one layer as its content layer.
- Other resource types MUST NOT have a content layer.

### 3.5. Paging Properties

**first**
A link from a resource with pages, such as a collection or layer, to its first page resource, another collection or an annotation list respectively. The page resource SHOULD be referenced by just its URI (from @id) but MAY also have more information associated with it as an object.

- A collection MAY have exactly one collection as its first page.
- A layer MAY have exactly one annotation list as its first page.
- Other resource types MUST NOT have a first page.

**last**
A link from a resource with pages to its last page resource. The page resource SHOULD be referenced by just its URI (from @id) but MAY also have more information associated with it as an object.

- A collection MAY have exactly one collection as its last page.
- A layer MAY have exactly one annotation list as its last page.
- Other resource types MUST NOT have a last page.

**total**
The total number of leaf resources, such as annotations within a layer, within a list of pages. The value MUST be a non-negative integer.

- A collection MAY have exactly one total, which MUST be the total number of collections and manifests in its list of pages.
- A layer MAY have exactly one total, which MUST be the total number of annotations in its list of pages.
- Other resource types MUST NOT have a total.

**next**
A link from a page resource to the next page resource that follows it in order. The resource SHOULD be referenced by just its URI (from @id) but MAY also have more information associated with it as an object.

- A collection MAY have exactly one collection as its next page.
- An annotation list MAY have exactly one annotation list as its next page.
- Other resource types MUST NOT have next pages.

**prev**
A link from a page resource to the previous page resource that precedes it in order. The resource SHOULD be referenced by just its URI (from @id) but MAY also have more information associated with it as an object.

- A collection MAY have exactly one collection as its previous page.
- An annotation list MAY have exactly one annotation list as its previous page.
- Other resource types MUST NOT have previous pages.
4. JSON-LD (JSON for Linking Data) Considerations

This section describes features applicable to all of the Presentation API (Application Programming Interface) content. For the most part, these are features of the JSON-LD (JSON for Linking Data) specification that have particular uses within the API (Application Programming Interface).

4.1. URI Representation

Resource descriptions SHOULD be embedded within higher-level descriptions, and MAY also be available via separate requests from http(s) URIs linked in the responses. These URIs are in the @id property for the resource. Links to resources MAY be either given as just the URI if there is no additional information associated with them, or as a JSON (JavaScript Object Notation) object with the @id property. Other URI schemes MAY be used if the resource is not able to be retrieved via HTTP (Hypertext Transfer Protocol). Both options provide the same URI, however the second pattern associates additional information with the resource:

- Option A, plain string
  
  ```json
  { "seeAlso": "http://example.org/descriptions/book1.xml" }
  ```

- Option B, object with @id property
  
  ```json
  { "seeAlso": { "@id": "http://example.org/descriptions/book1.xml", "format": "text/xml" }}
  ```

4.2. Repeated Properties

Many of the properties in the API (Application Programming Interface) MAY be repeated. This is done by giving a list of values, using either of the representations described above, rather than a single string.

```json
```

4.3. Language of PropertyValues

Language MAY be associated with strings that are intended to be displayed to the user with the following pattern of @value plus the RFC (Request for Comments) 5646 (http://tools.ietf.org/html/rfc5646) code in @language, instead of a plain string. For example:

```json
{ "description": { "@value": "Here is a longer description of the object", "@language": "en" }}
```

This pattern may be used in label, description, attribution and the label and value fields of the metadata construction.

Note that RFC (Request for Comments) 5646 (http://tools.ietf.org/html/rfc5646) allows the script of the text to be included after a hyphen, such as ar-latn, and clients should be aware of this possibility. This allows for full internationalization of the user interface components described in the response, as the labels as well as values may be translated in this manner; examples are given below.

In the case where multiple values are supplied, clients MUST use the following algorithm to determine which values to display to the user.

- If none of the values have a language associated with them, the client MUST display all of the values.
- Else, the client should try to determine the user’s language preferences, or failing that use some default language preferences. Then:
  - If any of the values have a language associated with them, the client MUST display all of the values associated with the language that best matches the language preference.
  - If all of the values have a language associated with them, and none match the language preference, the client MUST select a language and display all of the values associated with that language.
  - If some of the values have a language associated with them, but none match the language preference, the client MUST display all of the values that do not have a language associated with them.

4.4. HTML (HyperText Markup Language) Markup in Property Values

Minimal HTML (HyperText Markup Language) markup MAY be included in the description, attribution and metadata properties. It MUST NOT be used in label or other properties. This is included to allow manifest creators to add links and simple formatting instructions to blocks of text. The content MUST be well-formed XML (eXtensible Markup Language) and therefore must be wrapped in an element such as p or span. There MUST NOT be whitespace on either side of the HTML (HyperText Markup Language) string, and thus the first character in the
In order to avoid HTML (HyperText Markup Language) or script injection attacks, clients MUST remove:

- Tags such as `script`, `style`, `object`, `form`, `input` and similar.
- All attributes other than `href` on the `a` tag, `src` and `alt` on the `img` tag.
- CDATA sections.
- XML (eXtensible Markup Language) Comments.
- Processing instructions.

Clients SHOULD allow only `a`, `br`, `i`, `img`, `p`, and `span` tags. Clients MAY choose to remove any and all tags, therefore it SHOULD NOT be assumed that the formatting will always be rendered.

```json
{"description": {"@value": "Some <b>description</b>"}, "@language": "en-latn"}
```

## 4.5. Linked Data Context and Extensions

The top level resource in the response MUST have the `@context` property, and it SHOULD appear as the very first key/value pair of the JSON (JavaScript Object Notation) representation. This tells Linked Data processors how to interpret the information. The IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) context, below, MUST occur exactly once per response, and be omitted from any embedded resources. For example, when embedding a sequence within a manifest, the sequence MUST NOT have the `@context` field.

```json
{"@context": "http://iiif.io/api/presentation/2/context.json"}
```

Any additional fields beyond those defined in this specification SHOULD be mapped to RDF (Resource Description Framework) predicates using further context documents. In this case, the enclosing object MUST have its own `@context` property, and it SHOULD be the first key/value pair of that object. This is REQUIRED for service links that embed any information beyond a profile. These contexts SHOULD NOT redefine profile. As the top level resource MUST have the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) context, if there are any additional contexts needed, the value will become an array of URI strings:

```json
{
  "@context": [
    "http://iiif.io/api/presentation/2/context.json",
    "http://example.org/extension/context.json"
  ]
}
```

## 5. Resource Structure

This section provides detailed description of the resource types used in this specification. Section 2 provides an overview of the resource types and figures illustrating allowed relationships between them, and Appendix B provides summary tables of the property requirements.

### 5.1. Manifest

Recommended URI pattern:

```
{scheme}://{host}/{prefix}/{identifier}/manifest
```

The manifest response contains sufficient information for the client to initialize itself and begin to display something quickly to the user. The manifest resource represents a single object and any intellectual work or works embodied within that object. In particular it includes the descriptive, rights and linking information for the object. It then embeds the sequence(s) of canvases that should be rendered to the user.

The identifier in `@id` MUST always be able to be dereferenced to retrieve the JSON (JavaScript Object Notation) description of the manifest, and thus MUST use the http(s) URI scheme.

Along with the descriptive information, there is a `sequences` section, which is a list of JSON-LD (JSON for Linking Data) objects. Each object describes a Sequence, discussed in the next section, that represents the order of the parts of the work, each represented by a Canvas. The first such sequence MUST be included within the manifest as well as optionally being available from its own URI. Subsequent sequences MUST only be referenced with their identifier (`@id`), class (`@type`) and label and thus MUST be dereferenced by clients in order to process them if the user selects to view that sequence.

There MAY also be a `structures` section listing one or more Ranges which describe additional structure of the content, such as might be rendered as a table of contents.

The example below includes only the manifest-level information, however actual implementations MUST embed the first sequence, canvas and content information. It includes examples in the descriptive metadata of how to associate multiple entries with a single field and how to be explicit about the language of a particular entry.
5.2. Sequence

Recommended URI pattern:

{scheme}://{host}/prefix/identifier/sequence/name

The sequence conveys the ordering of the views of the object. The default sequence (and typically the only sequence) MUST be embedded within the manifest, and MAY also be available from its own URI. The default sequence MAY have a URI to identify it. Any additional sequences MUST be referred to from the manifest, not embedded within it, and thus these additional sequences MUST have an HTTP (Hypertext Transfer Protocol) URI.

The {name} parameter in the URI structure MUST distinguish it from any other sequences that may be available for the physical object. Typical default names for sequences are “normal” or “basic”.

Sequences MAY have their own descriptive, rights and linking metadata using the same fields as for manifests. The label property MAY be given for sequences and MUST be given if there is more than one referenced from a manifest. After the metadata, the set of pages in the object, represented by canvas resources, are listed in order in the canvases property. There MUST be at least one canvas given.

Sequences MAY have a startCanvas with a single value containing the URI of a canvas resource that is contained within the sequence. This is the canvas that a viewer SHOULD initialize its display with for the user. If it is not present, then the viewer SHOULD use the first canvas in the sequence.

In the manifest example above, the sequence is referenced by its URI and contains only the basic information of label, @type and @id. The default sequence should be written out in full within the manifest file, as below but MUST NOT have the @context property.

```
{
    // Metadata about this sequence
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/sequence/normal",
    "@type": "sc:Sequence",
    "label": "Current Page Order",
    "viewingDirection": "left-to-right",
    "viewingHint": "paged",
    "startCanvas": "http://example.org/iiif/book1/canvas/p2",

    // The order of the canvases
    "canvases": [
        {
            "@id": "http://example.org/iiif/book1/canvas/p1",
            "@type": "sc:Canvas",
            "label": "p. 1"
        },
        {
            "@id": "http://example.org/iiif/book1/canvas/p2",
            "@type": "sc:Canvas",
            "label": "p. 2"
        },
        {
            "@id": "http://example.org/iiif/book1/canvas/p3",
            "@type": "sc:Canvas",
            "label": "p. 3"
        }
    ]
}
```

5.3. Canvas

Recommended URI pattern:

{scheme}://{host}/prefix/identifier/canvas/name

The canvas represents an individual page or view and acts as a central point for laying out the different content resources that make up the display. Canvases MUST be identified by a URI and it MUST be an HTTP (Hypertext Transfer Protocol) URI. If following the recommended URI pattern, the {name} parameter MUST uniquely distinguish the canvas from all other canvases in the object. The URI of the canvas SHOULD NOT contain a fragment (a # followed by further characters), as this would make it impossible to refer to a segment of the canvas's area using the #xywh= syntax.
Every canvas MUST have a **label** to display, and a **height** and a **width** as integers. A canvas is a two-dimensional rectangular space with an aspect ratio that represents a single logical view of some part of the object, and the aspect ratio is given with the height and width properties. This allows resources to be associated with specific parts of the canvas, rather than the entire space. Content MUST NOT be associated with space outside of the canvas’s dimensions, such as at coordinates below 0.0 or greater than the height or width.

It is **RECOMMENDED** that if there is (at the time of implementation) a single image that depicts the page, then the dimensions of the image are used as the dimensions of the canvas for simplicity. If there are multiple full images, then the dimensions of the largest image should be used. If the largest image’s dimensions are less than 1200 pixels on either edge, then the canvas’s dimensions SHOULD be double those of the image. Clients MUST be aware that this is not always the case, such as in the examples presented, and instead MUST always scale images into the space represented by the canvas. The dimensions of the canvas SHOULD be the same scale as the physical object, and thus images SHOULD depict only the object. This can be accomplished by cropping the image, or associating only a segment of the image with the canvas. The physical dimensions of the object may be available via a service, either embedded within the description or requiring an HTTP (Hypertext Transfer Protocol) request to retrieve them.

Image resources, and only image resources, are included in the **images** property of the canvas. These are linked to the canvas via annotations, as described in Image Resources. Other content, such as transcriptions, video, audio or commentary, is provided via external annotation lists referenced in the **otherContent** property, as described in Annotation Lists. The value of both of these properties MUST be a list, even if there is only one entry. Both are optional, as there may be no additional information associated with the canvas. Note that the items in the **otherContent** list may be either objects with an **@id** property or strings. In the case of a string, this is the URI of the annotation list and the type of “sc:AnnotationList” can be inferred.

In a sequence with the **viewingHint** value of “paged” and presented in a book viewing user interface, the first canvas SHOULD be presented by itself – it is typically either the cover or first recto page. Thereafter, the canvases represent the sides of the leaves, and hence may be presented with two canvases displayed as an opening of the book. If there are canvases which are in the sequence but would break this ordering, then they MUST have the **viewingHint** property with a value of “non-paged”. Similarly if the first canvas is not a single up, it MUST be marked as “non-paged” or an empty canvas added before it.

Canvases MAY be dereferenced separately from the manifest via their URIs, and the following representation information should be returned. This information should be embedded within the sequence, as per previously.

```json
{
   // Metadata about this canvas
   "@context": "http://iiif.io/api/presentation/2/context.json",
   "@id": "http://example.org/iiif/book1/canvas/p1",
   "@type": "sc:Canvas",
   "label": "p. 1",
   "height": 1000,
   "width": 750,
   "thumbnail": {
      "@id": "http://example.org/iiif/book1/canvas/p1/thumb.jpg",
      "@type": "dctypes:Image",
      "height": 200,
      "width": 150
   },
   "images": [
      {
         "@type": "oa:Annotation"
      }],
   "otherContent": [
      {
         "@id": "http://example.org/iiif/book1/list/p1",
         "@type": "sc:AnnotationList"
      }
   ]
}
```

### 5.4 Image Resources

**Recommended URI pattern:**

```
{scheme}:://{host}/{prefix}/{identifier}/annotation/{name}
```

Association of images with their respective canvases is done via annotations. Although normally annotations are used for associating commentary with the thing the annotation’s text is about, the Open Annotation (http://www.openannotation.org/spec/core/) model allows any resource to be associated with any other resource, or parts thereof, and it is reused for both commentary and painting resources on the canvas.

Annotations MAY have their own URIs, conveyed by adding an **@id** property to the JSON (JavaScript Object Notation) object, and if so SHOULD be HTTP (Hypertext Transfer Protocol) URIs. The content of the annotation SHOULD be returned if the URI is dereferenced. Annotations MAY be dereferenced separately from their annotation lists, sequences and manifests; some systems may do this and identifiers should be given using the recommended pattern if possible.
Each association of an image MUST have the motivation field and the value MUST be "sc:painting". This is in order to distinguish it from comment annotations about the canvas, described in further detail below. Note that all resources which are to be displayed as part of the representation are given the motivation of "sc:painting", regardless of whether they are images or not. For example, a transcription of the text in a page is considered "painting" as it is a representation of the object, whereas a comment about the page is not.

The image itself is linked in the resource property of the annotation. The image MUST have an @id field, with the value being the URI at which the image can be obtained. If a IIIF (International Image Interoperability Framework) Image service is available for the image, then the URL MAY be the complete URL to a particular size of the image content, such as http://example.org/image1/full/1000,0/default.jpg. It SHOULD have an @type of "dctypes:Image". Its media type MAY be listed in format, and its height and width MAY be given as integer values for height and width respectively.

If a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service is available for the image, then a link to the service's base URI SHOULD be included. The base URI is the URI up to the identifier, but not including the trailing slash character or any of the subsequent parameters. A reference to the Image API (Application Programming Interface) context document MUST be included and the conformance level profile of the service SHOULD be included. Additional fields from the Image Information document MAY be included in this JSON (JavaScript Object Notation) object to avoid requiring it to be downloaded separately. See the annex (/api/annex/services/) on using external services for more information.

Although it seems redundant, the URI of the canvas MUST be repeated in the on field of the Annotation. This is to ensure consistency with annotations that target only part of the resource, described in more detail below.

Additional features of the Open Annotation (http://www.openannotation.org/spec/core/) data model MAY also be used, such as selecting a segment of the canvas or content resource, or embedding the comment or transcription within the annotation. These additional features are described in the following section. The use of advanced features sometimes results in situations where the resource is not an image, but instead a SpecificResource, a choice or other non content object. Implementations should check the type of the resource and not assume that it is always an image.

Only the annotations that associate images or parts of images are included in the canvas in the images property. Other annotations, including those that paint resources on the canvas and those that comment about the canvas, are included by referencing annotation lists, discussed in the following section.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/annotation/p0001-image",
    "@type": "oa:Annotation",
    "motivation": "sc:painting",
    "resource": {
        "@id": "http://example.org/iiif/book1/res/page1.jpg",
        "@type": "dctypes:Image",
        "format": "image/jpeg",
        "service": {
            "@context": "http://iiif.io/api/image/2/context.json",
            "@id": "http://example.org/iiif/book1/res/page1.jpg",
            "profile": "http://iiif.io/api/image/2/profile.json"
        }
    },
    "height": 2000,
    "width": 1500,
    "on": "http://example.org/iiif/book1/annotation/p0001-image"
}
```

5.5. Annotation List

Recommended URI pattern:

```
{scheme}:://{host}/{prefix}/{identifier}/list/{name}
```

For some objects, there may be more than just images available to represent the page. Other resources could include the full text of the object, musical notations, musical performances, diagram transcriptions, commentary annotations, tags, video, data and more. These additional resources are included in annotation lists, referenced from the canvas they are associated with.

Annotation Lists are separate resources that SHOULD be dereferenced when encountered. They are collections of annotations, where each annotation targets the Canvas or part thereof. The separation from the manifest representation is intended to allow clients to quickly display the images to the user, and then populate the display with further content and commentary when the user navigates to a particular canvas. It also allows the annotation list to be generated dynamically, while the manifest is static and more easily cached.

The {name} parameter in the URI pattern MUST uniquely distinguish it from all other lists, and is typically the same name as the canvas. As a single canvas may have multiple lists of additional resources, perhaps divided by type, this MUST NOT be assumed however, and the URIs must be followed rather than constructed a priori.

The annotation list MUST have an http(s) URI given in @id, and the JSON (JavaScript Object Notation) representation MUST be returned when that URI is dereferenced. They MAY have any of the other fields defined in this specification.

The annotations, as described above, are given in a resources list. The resource linked by the annotation MUST be something other than an image if the motivation is sc:painting, these are recorded in the images property of the canvas. The canvas URI MUST be repeated in the on field, as above.

The format of the resource SHOULD be included and MUST be the media type that is returned when the resource is dereferenced. The type of the content resource SHOULD be taken from this list in the Open Annotation specification (http://www.openannotation.org/spec/core/core.html#BodyTargetType), or a similar well-known resource type ontology. For resources that are
displayed as part of the rendering (such as images, text transcriptions, performances of music from the manuscript and so forth) the motivation MUST be "sc:painting". The content resources MAY also have any of the other fields defined in this specification, including commonly label, description, metadata, license and attribution.

Note well that Annotation Lists MUST NOT be embedded within the manifest.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/list/p1",
    "@type": "sc:AnnotationList",
    "resources": [
        {
            "@type": "oa:Annotation",
            "motivation": "sc:painting",
            "resource": {
                "@id": "http://example.org/iiif/book1/res/music.mp3",
                "@type": "dctypes:Sound",
                "format": "audio/mpeg"
            },
            "on": "http://example.org/iiif/book1/canvas/p1"
        },
        {
            "@type": "oa:Annotation",
            "motivation": "sc:painting",
            "resource": {
                "@id": "http://example.org/iiif/book1/res/tei-text-p1.xml",
                "@type": "dctypes:Text",
                "format": "application/tei+xml"
            },
            "on": "http://example.org/iiif/book1/canvas/p1"
        }
    // ... and so on
}
```

5.6. Range

Recommended URI pattern:

```
{scheme}://{host}/{prefix}/{identifier}/range/{name}
```

It may be important to describe additional structure within an object, such as newspaper articles that span pages, the range of non-content-bearing pages at the beginning of a work, or chapters within a book. These are described using ranges in a similar manner to sequences. Ranges MUST have URIs and they SHOULD be http(s) URIs. The intent of adding a range to the manifest is to allow the client to display a structured hierarchy to enable the user to navigate within the object without merely stepping through the current sequence. The rationale for separating ranges from sequences is that there is likely to be overlap between different ranges, such as the physical structure of a book compared to the textual structure of the work. An example would be a newspaper with articles that are continued in different sections, or simply a section that starts half way through a page.

Ranges are linked or embedded within the manifest in a structures field. It is a flat list of objects, even if there is only one range.

Ranges have three list based properties to express membership:

- **ranges**
  References to ranges within the current range. Each included range MUST be referenced via a string containing the range’s URI.

- **canvases**
  References to canvases, or rectangular parts of a canvas, within the current range. Each included canvas MUST be referenced via a string containing the canvas’s URI.

- **members**
  A combined list of both ranges and canvases. If the range contains both other ranges and canvases, and the ordering of the different types of resource is significant, the range SHOULD instead use the members property. The property’s value is an array of canvases, parts of canvases or other ranges. Each item in the array MUST be an object, and it MUST have the @id, @type, and label properties.

A range will typically include one or more canvases or, unlike sequences, parts of canvases. The part must be rectangular, and is given using the xywh fragment approach. This allows for selecting, for example, the areas within two newspaper pages where an article is located. An empty range, with no member resources, is allowed but discouraged. The reason for the empty range could be described in the label property, or in the description property for more discursive text.

In order to present a table of the different ranges to allow a user to select one, every range MUST have a label and the top most range in the table SHOULD have a viewingHint with the value “top”. A range that is the top of a hierarchy does not need to list all of the canvases in the sequence, and SHOULD only give the list of ranges below it. Ranges MAY also have any of the other properties defined in this specification, including the startCanvas relationship to the first canvas within the range to start with, if it is not the first listed in canvases or members.

Ranges MAY also link to a layer, described in the next section, that has the content of the range using the contentLayer linking property. The referenced layer will contain one or more annotation lists, each of which contains annotations that target the areas of canvases within the range, and provide the content resources. This allows, for example, the range representing a newspaper article that is split across multiple
Deprecation Warning  Several issues have arisen with respect to the current specification for ranges, and a new pattern is anticipated in API (Application Programming Interface) version 3.0 to address these concerns. Feedback on this deprecation is requested (mailto:iiif-discuss@googlegroups.com).

5.7. Layer

Recommended URI pattern:

    {scheme}://{host}/{prefix}/{identifier}/layer/{name}

Layers represent groupings of annotation lists that should be collected together, regardless of which canvas they target, such as all of the annotations that make up a particular translation of the text of a book. Without the layer construction, it would be impossible to determine which annotations belonged together across canvases. A client might then present a user interface that allows all of the annotations in a layer to be displayed or hidden according to the user’s preference.

Layers MUST have a URI, and it SHOULD be an HTTP (Hypertext Transfer Protocol) URI. They MUST have a label and MAY have any of the other descriptive, linking or rights properties.
Each annotation list MAY be part of one or more layers. If the annotation list is part of a layer, then this MUST be recorded using the within relationship in the annotation list response. It MAY also be included in the reference to the annotation list in the manifest response. In the manifest response, the description of the layer MAY be omitted after the first use, and just the URI given as a string. Clients should refer to the first description given, based on the URI.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/list/l1",
    "@type": "sc:AnnotationList",
    "within": {
        "@id": "http://example.org/iiif/book1/layer/transcription",
        "@type": "sc:Layer",
        "label": "Diplomatic Transcription"
    }
}
```

The layer MAY be able to be dereferenced if it has an HTTP (Hypertext Transfer Protocol) URI. If a representation is available, it MUST follow all of the requirements for JSON (JavaScript Object Notation) representations in this specification. All of the properties of the layer SHOULD be included in the representation.

The annotation lists are referenced from the layer in an otherContent array, in the same way as they are referenced from a canvas. The annotation lists SHOULD be given as just URIs, but MAY be objects with more information about them, such as in the Canvas example.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/layer/transcription",
    "@type": "sc:Layer",
    "label": "Diplomatic Transcription",
    // Other properties here ...

    "otherContent": [
        "http://example.org/iiif/book1/list/l1",
        "http://example.org/iiif/book1/list/l2",
        "http://example.org/iiif/book1/list/l3",
        "http://example.org/iiif/book1/list/l4"
        // More AnnotationLists here ...
    ]
}
```

5.8. Collection

Recommended URI pattern:

```
{scheme}://{host}/{prefix}/collection/{name}
```

Collections are used to list the manifests available for viewing, and to describe the structures, hierarchies or curated collections that the physical objects are part of. The collections MAY include both other collections and manifests, in order to form a hierarchy of objects with manifests at the leaf nodes of the tree. Collection objects MAY be embedded inline within other collection objects, such as when the collection is used primarily to subdivide a larger one into more manageable pieces, however manifests MUST NOT be embedded within collections. An embedded collection SHOULD also have its own URI from which the description is available.

The URI pattern follows the same structure as the other resource types, however note that it prevents the existence of a manifest or object with the identifier "collection". It is also RECOMMENDED that the topmost collection from which all other collections are discoverable by following links within the hierarchy be named top, if there is one.

Manifests or collections MAY appear within more than one collection. For example, an institution might define four collections: one for modern works, one for historical works, one for newspapers and one for books. The manifest for a modern newspaper would then appear in both the modern collection and the newspaper collection. Alternatively, the institution may choose to have two separate newspaper collections, and reference each as a sub-collection of modern and historical.

The intended usage of collections is to allow clients to:

- Load a pre-defined set of manifests at initialization time.
- Receive a set of manifests, such as search results, for rendering.
- Visualize lists or hierarchies of related manifests.
- Provide navigation through a list or hierarchy of available manifests.

As such, collections MUST have a label, and SHOULD have metadata and description properties to be displayed by the client such that the user can understand the structure they are interacting with. If a collection does not have these properties, then a client is not required to render the collection to the user directly.

Collections have three list-based properties to express membership:

- **collections**
  - References to sub-collections of the current collection. Each referenced collection MUST have the appropriate @id, @type and label, and MAY be embedded in its entirety.

- **manifests**
References to manifests contained within the current collection. Each referenced manifest MUST have the appropriate @id, @type and label.

members

In cases where the order of a collection is significant, members can be used to interleave both collection and manifest resources. This is especially useful when a collection of books contains single- and multi-volume works (i.e. collections with the "multi-part" viewingHint), and when modeling archival material where original order is significant. Each entry in the members list MUST be an object and MUST include @id, @type, and label. If the entry is a collection, then viewingHint MUST also be present.

At least one of collections, manifests and members SHOULD be present in the response. An empty collection, with no member resources, is allowed but discouraged.

An example collection document:

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/collection/top",
    "@type": "sc:Collection",
    "label": "Top Level Collection for Example Organization",
    "viewingHint": "top",
    "description": "Description of Collection",
    "attribution": "Provided by Example Organization",
    "collections": [
        {
            "@id": "http://example.org/iiif/collection/sub1",
            "@type": "sc:Collection",
            "label": "Sub-Collection 1",
            "members": [
                {
                    "@id": "http://example.org/iiif/collection/part1",
                    "@type": "sc:Collection",
                    "label": "My Multi-volume Set",
                    "viewingHint": "multi-part"
                },
                {
                    "@id": "http://example.org/iiif/book1/manifest1",
                    "@type": "sc:Manifest",
                    "label": "My Book"
                },
                {
                    "@id": "http://example.org/iiif/collection/part2",
                    "@type": "sc:Collection",
                    "label": "My Sub Collection",
                    "viewingHint": "individuals"
                }
            ]
        },
        {
            "@id": "http://example.org/iiif/collection/part2",
            "@type": "sc:Collection",
            "label": "Sub Collection 2"
        }
    ],
    "manifests": [
        {
            "@id": "http://example.org/iiif/book1/manifest",
            "@type": "sc:Manifest",
            "label": "Book 1"
        }
    ]
}
```

Deprecation Warning  The collections and manifests properties are likely to be removed in version 3.0 in favor of the single members property. Until that time, if a client sees a members property, it should use that property even if collections and/or manifests are also present. However, publishing systems should be aware that Presentation API (Application Programming Interface) version 2.0-compliant clients will not produce the expected results if they use members and do not provide a fallback with collections and manifests. Publishing systems should only use members when it is important to have a single ordered list that contains both collections and manifests. Feedback on this deprecation is requested (mailto:iiif-discuss@googlegroups.com).

5.9. Paging
In some situations, annotation lists or the list of manifests in a collection may be very long or expensive to create. The latter case is especially likely to occur when responses are generated dynamically. In these situations the server may break up the response using paging properties.

The length of a response is left to the server’s discretion, but the server should take care not to produce overly long responses that would be difficult for clients to process.

When breaking a response into pages, the paged resource MUST link to the first page resource, and MUST NOT include the corresponding list property (collections for a collection, otherContent for a layer). For example, a layered would link only to an annotation list as its first page. If known, the resource MAY also link to the last page.

The linked page resource SHOULD refer back to the containing paged resource using within. If there is a page resource that follows it (the next page), then it MUST include a next link to it. If there is a preceding page resource, then it SHOULD include a prev link to it.

The paged resource MAY use the total property to list the total number of leaf resources that are contained within its pages. This would be the total number of annotations in a layer, or the total number of manifests in a collection. Conversely, the page resources MAY include the startIndex property with index of the first resource in the page, counting from zero relative to the containing paged resource.

The linked page resources MAY have different properties from the paged resource, including different rights and descriptive properties. Clients MUST take into account any requirements derived from these properties, such as displaying logo or attribution.

**Example Paged Layer**

A layer representing a long transcription with almost half a million annotations, perhaps where each annotation paints a single word on the canvas:

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/layer/transcription",
    "@type": "sc:Layer",
    "label": "Example Long Transcription",
    "total": 496923,
    "first": "http://example.org/iiif/book1/list/l1"
}
```

And the corresponding first annotation list:

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/list/l1",
    "@type": "sc:AnnotationList",
    "startIndex": 0,
    "within": "http://example.org/iiif/book1/layer/transcription",
    "next": "http://example.org/iiif/book1/list/l2",
    "resources": [
        // Annotations live here ...
    ]
}
```

Note that it is still expected that canvases will link directly to the annotation lists. For example, a particular canvas might refer to the first two annotation lists within a layer:

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/canvas/c1",
    "@type": "sc:Canvas",
    "height": 1000,
    "width": 1000,
    "label": "Page 1",
    "otherContent": [
        "http://example.org/iiif/book1/list/l1",
        "http://example.org/iiif/book1/list/l2"
    ]
}
```

**Example Paged Collection**

An example large collection with some 9.3 million objects in it:
6. Advanced Association Features

The following sections describe known use cases for building representations of objects using the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface), and clients SHOULD expect to encounter them. Other use cases are likely to exist, and MUST be encoded using the Open Annotation’s (http://www.openannotation.org/spec/core/) context document mapping for any additional fields required.

6.1. Segments

It is important to be able to extract parts, or segments, of resources. In particular a very common requirement is to associate a resource with part of a canvas, or part of an image with either the entire canvas or part thereof. Secondly, as transcriptions are often made available in XML (eXtensible Markup Language) files, extracting the correct page to associate with the canvas, or line to associate with part of the canvas, is equally useful for reusing existing material. These can be accomplished using URI fragments for simple cases.

Note that if there are segments of both image and canvas, then the aspect ratio SHOULD be the same, but there are circumstances where they MAY be different. In this case the rendering agent SHOULD rescale the image segment to the dimensions provided on the canvas.

Segments of both static images and canvases may be selected by adding a rectangular bounding box (http://www.w3.org/TR/media-frags/#naming-space) after the URI. The fragment MUST take the form of \#xywh= as per the example below where the four numbers are the x and y coordinates of the top left hand corner of the bounding box in the image or canvas, followed by the width and height. Thus the segment above is 300px wide, 50px high and starts at position 100,100. Note that only integers are allowed in this syntax, and this may limit accuracy of assignment to canvases with small dimensions.

```
http://www.example.com/iiif/book1/canvas/p1#xywh=100,100,300,50
```

For image resources with a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service, it is RECOMMENDED to instead use the Image API (Application Programming Interface) parameters rather than a fragment as above. The following structure allows simple clients to use the image directly (the URL with the segment), and allows clients that implement the IIIF Image API (Application Programming Interface) to have sufficient information to construct appropriate URIs using the API (Application Programming Interface).
Segments of XML (eXtensible Markup Language) files may be extracted with XPaths (https://en.wikipedia.org/wiki/XPointer). The fragment MUST be structured as follows:

```
http://www.example.com/iiif/book1/res/tei.xml#xpointer(/xpath/to/element)
```

6.2. Embedded Content

Instead of referencing transcription text externally, it is often easier to record it within the annotation itself. Equally, text based comments could also benefit from being included in the annotation that associates the comment with the canvas.

Content MAY be embedded instead of referenced by using the following pattern within the annotation block:

```
{"resource": {"@type": "cnt:ContentAsText", "chars": "text here"}}
```

The media type SHOULd be provided using the format field, and while any media type is possible, it is RECOMMENDED that text/plain or text/html be used to maximize compatibility.

If it is desirable to describe the language of the content, then it MUST be given with the language property not @language.

An example of this feature:

```
{"@context": "http://iiif.io/api/presentation/2/context.json",
"@id": "http://example.org/iiif/book1/annotation/p1",
"@type": "oa:Annotation",
"motivation": "sc:painting",
"resource": {
  "@type": "cnt:ContentAsText",
  "chars": "Here starts book one...",
  "format": "text/plain",
  "language": "en"
},
"on": "http://example.org/iiif/book1/canvas/p1#xywh=100,150,500,25"
}
```

6.3. Choice of Alternative Resources
A common requirement is to have a choice between multiple images that depict the page, such as being photographed under different lights or at different times. This can be accomplished by having a “oa:Choice” object as the resource, which then refers to the options to select from. It MUST have one default and at least one further item to choose from. The images SHOULD have a label for the viewer to display to the user so they can make their selection from among the options.

The same construction can be applied to a choice between other types of resources as well. This is described in the Multiplicity section (http://www.openannotation.org/spec/core/multiplicity.html#Choice) of the Open Annotation specification.

Either the default or item MAY have a value of “rdf:nil”. This means that a valid option is not to display anything. This MUST NOT have a label associated with it, viewers should either use “Nothing” or an appropriate label of their choice.

This can be used to model foldouts and other dynamic features of a page, by associating images of the different states with the canvas. Depending on the nature of the images, this can be done such that either the entire image is switched to change state, or only the section of the image that has to change is switched, if the appropriate segment information is known.

```json
{
  "@context": "http://iiif.io/api/presentation/2/context.json",
  "@id": "http://example.org/iiif/book1/annotation/anno1",
  "@type": "oa:Annotation",
  "motivation": "sc:painting",
  "resource": {
    "@type": "oa:Choice",
    "default": {
      "@id": "http://example.org/iiif/book1/res/page1.jpg",
      "@type": "dctypes:Image",
      "label": "Color"
    },
    "item": [
      {
        "@id": "http://example.org/iiif/book1/res/page1-blackandwhite.jpg",
        "@type": "dctypes:Image",
        "label": "Black and White"
      }
    ],
    "on": "http://example.org/iiif/book1/canvas/p1"
  }
}
```

### 6.4. Non Rectangular Segments

The Scalable Vector Graphics (http://www.w3.org/TR/SVG/) standard (SVG) is used to describe non-rectangular, and rotated rectangular, areas of canvas or image resources. In this pattern, the resource of the annotation is a “oa:SpecificResource” which has the complete image referenced in a full field and the SVG embedded in a selector field (as the SVG selects the part of the image needed). The SVG document is embedded using the same ContentAsText approach as for embedding comments or transcriptions.

If the section of an image is mapped to part of a canvas, as in the example below, then the target in on MUST be the rectangular bounding box in which the SVG viewport should be placed. If the entire canvas is the target, then the SVG viewport is assumed to cover the entire canvas. If the dimensions of the viewport and the bounding box or canvas are not the same, then the SVG MUST be scaled such that it covers the region. This may result in different scaling ratios for the X and Y dimensions.

SVG SHOULD NOT be used to describe non-rotated rectangular regions. The IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) or the `xywh` bounding box described above SHOULD be used instead.

```json
{
  "@context": "http://iiif.io/api/presentation/2/context.json",
  "@id": "http://example.org/iiif/book1/annotation/anno1",
  "@type": "oa:Annotation",
  "motivation": "sc:painting",
  "resource": {
    "@type": "oa:SpecificResource",
    "full": {
      "@id": "http://example.org/iiif/book1/res/page1.jpg",
      "@type": "dctypes:Image"
    },
    "selector": {
      "@type": "oa:SvgSelector","cnt:ContentAsText",
      "chars": "<svg xmlns="..." path d="..."/>"
    }
  },
  "on": "http://example.org/iiif/book1/canvas/p1#xywh=100,100,300,300"
}
```

### 6.5. Style

The Cascading Style Sheets (http://www.w3.org/TR/CSS/) standard (CSS) is used to describe how the client should render a given resource to the user. The CSS information is embedded within the annotation using the same ContentAsText approach above. As a stylesheet may contain more than one style, and be reused between annotations, it is attached to the annotation directly in the same manner as a stylesheet.
being linked to an HTML (HyperText Markup Language) document. Then the name of the style class is attached to the resource that should be styled, again in the same manner as the class attribute in html, although we use style to avoid confusion with object classes.

In the example below, the text should be colored red.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/annotation/anno1",
    "@type": "oa:Annotation",
    "motivation": "sc:painting",
    "stylesheet":{
        "@type": ["oa:CssStyle", "cnt:ContentAsText"],
        "chars": ".red {color: red;}
    },
    "resource":{
        "@type": "oa:SpecificResource",
        "style": "red",
        "full":{
            "@type": "cnt:ContentAsText",
            "chars": "Rubrics are Red, ..."
        }
    },
    "on": "http://example.org/iiif/book1/canvas/p1#xywh=100,150,500,30"
}
```

6.6. Rotation

CSS may also be used for rotation of images which are not correctly aligned with the canvas. In the example below, after the image is located within the 500 wide by 30 high space within the canvas, it is then rotated by the rendering client application around the top left corner by 45 degrees anti-clockwise.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/annotation/anno1",
    "@type": "oa:Annotation",
    "motivation": "sc:painting",
    "stylesheet":{
        "@type": ["oa:CssStyle", "cnt:ContentAsText"],
        "chars": ".rotated {transform-origin: top left; transform: rotate(-45deg);}
    },
    "resource":{
        "@type": "oa:SpecificResource",
        "style": "rotated",
        "full": {
            "@id": "http://example.org/iiif/book1/res/page1-detail.png",
            "@type": "dctypes:Image"
        }
    },
    "on": "http://example.org/iiif/book1/canvas/p1#xywh=100,150,500,30"
}
```

Alternatively, if the image is available via the IIIF (International Image Interoperability Framework) Image API (Application Programming Interface), it may be more convenient to have the server do the rotation of the image. This uses a custom Selector for the Image API (Application Programming Interface), further described in the Open Annotation extensions (/api/annex/openannotation/) annex. For the purposes of rotation, the example below demonstrates the pattern.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/iiif/book1/annotation/anno1",
    "@type": "oa:Annotation",
    "motivation": "sc:painting",
    "stylesheet":{
        "@type": ["oa:CssStyle", "cnt:ContentAsText"],
        "chars": ".rotated {transform-origin: top left; transform: rotate(-45deg);}
    },
    "resource":{
        "@type": "oa:SpecificResource",
        "style": "rotated",
        "full": {
            "@id": "http://example.org/iiif/book1/res/page1-detail.png",
            "@type": "dctypes:Image"
        }
    },
    "on": "http://example.org/iiif/book1/canvas/p1#xywh=100,150,500,30"
}
```
6.7. Comment Annotations

For annotations which are comments about the canvas, as opposed to painting content resources onto the canvas, there are different types of motivation to make the distinction clear. For annotations about the content (such as comments, notes, descriptions etc.) the motivation SHOULD be "oa:commenting", but MAY be any from the list given in the Open Annotation (http://www.openannotation.org/spec/core/) specification.

Unlike painting annotations, comments or annotations with other motivations SHOULD have a URI assigned as their identity and provided in the @id property. When dereferencing that URI, the representation of the annotation SHOULD be returned. This is to allow further annotations to annotate the comment, for example in order to reply to it, or to tag it for organizational or discovery purposes.

Other resources may also have comments made about them, including manifests (comments about the object), sequences (comments about that particular ordering), ranges (comments about the section), annotations (replies to the targeted annotation), and so forth. In order for the client to discover these annotations, they can be included in an AnnotationList referenced from the target resource. This is accomplished by reusing the otherContent pattern. Any resource may have a list of annotations associated with it in this way.

6.8. Hotspot Linking

It is also possible to use annotations to create links between resources, both within the manifest or to external content. This can be used to link to the continuation of an article in a digitized newspaper in a different canvas, or to link to an external web page that describes the diagram in the canvas.
Hotspot linking is accomplished using an annotation with a motivation of "oa:linking". The region of the canvas that should trigger the link when clicked is specified in the on field in the same way as other annotations. The linked resource is given in the resource field. The linked resource MAY also be another canvas or region of a canvas. The user experience of whether the linked resource is opened in a new tab, new window or by replacing the current view is up to the implementation.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://www.example.org/iiif/book1/annotation/anno1",
    "@type": "oa:Annotation",
    "motivation": "oa:linking",
    "resource": {
        "@id": "http://www.example.org/page-to-go-to.html",
        "@type": "dctypes:Text",
        "format": "text/html"
    },
    "on": "http://www.example.org/iiif/book1/canvas/p1#xywh=500,500,150,30"
}
```

7. HTTP (Hypertext Transfer Protocol) Requests and Responses

This section describes the RECOMMENDED request and response interactions for the API (Application Programming Interface). The REST (Representational State Transfer) and simple HATEOAS (Hypermedia as the Engine of Application State) approach is followed where an interaction will retrieve a description of the resource, and additional calls may be made by following links obtained from within the description. All of the requests use the HTTP (Hypertext Transfer Protocol) GET method; creation and update of resources is not covered by this specification.

7.1. Requests

Each of the entries in section 4 recommends a URI pattern to follow for the different resources. Following these patterns is NOT REQUIRED and clients MUST NOT construct the URIs by themselves, instead they MUST follow links from within retrieved descriptions.

The Base URI, to which additional information is appended, that is RECOMMENDED for resources made available by the API (Application Programming Interface) is:

```
{scheme}://{host}{/prefix}/{identifier}
```

Where the parameters are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>Indicates the use of the http or https protocol in calling the service.</td>
</tr>
<tr>
<td>server</td>
<td>The host server (and optional port) on which the service resides.</td>
</tr>
<tr>
<td>prefix</td>
<td>The path on the host server to the service. This prefix is optional, but may</td>
</tr>
<tr>
<td></td>
<td>be useful when the host server supports multiple services. The prefix MAY</td>
</tr>
<tr>
<td></td>
<td>contain multiple path segments, delimited by slashes, but all other special</td>
</tr>
<tr>
<td></td>
<td>characters MUST be encoded.</td>
</tr>
<tr>
<td>identifier</td>
<td>The identifier for the object or collection, expressed as a string. This</td>
</tr>
<tr>
<td></td>
<td>may be an ark, URN, or other identifier. Special characters MUST be URI</td>
</tr>
<tr>
<td></td>
<td>encoded.</td>
</tr>
</tbody>
</table>

The individual resources SHOULD have URIs below this top-level pattern, formed by appending a "?" and additional information to identify the resource. Recommended patterns for these URIs are given in the sections below for the different resource types, and summarized in Appendix A.

In the situation where the JSON (JavaScript Object Notation) documents are maintained in a filesystem with no access to the web server’s configuration, then including "json" on the end of the URI is suggested to ensure that the correct content-type response header is sent to the client. While this does not follow the recommended URI patterns below, it is not prevented by the specification either.

7.2. Responses

The format for all responses is JSON (JavaScript Object Notation), and the following sections describe the structure to be returned.

The content-type of the response MUST be either application/json (regular JSON (JavaScript Object Notation)),

```
Content-Type: application/json
```

or "application/ld+json" (JSON-LD (JSON for Linking Data)).

```
Content-Type: application/ld+json
```

If the client explicitly wants the JSON-LD (JSON for Linking Data) content-type, then it MUST specify this in an Accept header, otherwise the server MUST return the regular JSON (JavaScript Object Notation) content-type.

The HTTP (Hypertext Transfer Protocol) server SHOULD, if at all possible, send the Cross Origin Access Control header (often called "CORS (Cross-Origin Resource Sharing)") to allow clients to download the manifests via AJAX from remote sites. The header name is Access-Control-Allow-Origin and the value of the header SHOULD be *.
Responses SHOULD be compressed by the server as there are significant performance gains to be made for very repetitive data structures. Recipes for enabling CORS (Cross-Origin Resource Sharing) and the conditional Content-type header are provided in the Apache HTTP (Hypertext Transfer Protocol) Server Implementation Notes (/api/annex/notes/apache/).

8. Authentication

It may be necessary to restrict access to the descriptions made available via the Presentation API (Application Programming Interface). As the primary means of interaction with the descriptions is by web browsers using XMLHttpRequests across domains, there are some considerations regarding the most appropriate methods for authenticating users and authorizing their access. The approach taken is described in the Authentication (api/auth/) specification, and requires requesting a token to add to the requests to identify the user. This token might also be used for other requests defined by other APIs.

It is possible to include Image API (Application Programming Interface) service descriptions within the manifest, and within those it is also possible to include links to the Authentication API (Application Programming Interface)’s services that are needed to interact with the image content. The first time an Authentication API (Application Programming Interface) service is included within a manifest, it MUST be the complete description. Subsequent references SHOULD be just the URL of the service, and clients are expected to look up the details from the full description by matching the URL. Clients MUST anticipate situations where the Authentication service description in the manifest is out of date: the source of truth is the Image Information document, or other system that references the Authentication API (Application Programming Interface) services.

Appendices

A. Summary of Recommended URI Patterns

<table>
<thead>
<tr>
<th>Resource</th>
<th>URI Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>{scheme}://(host)&gt;({prefix}/collection/(name)</td>
</tr>
<tr>
<td>Manifest</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/manifest</td>
</tr>
<tr>
<td>Sequence</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/sequence/(name)</td>
</tr>
<tr>
<td>Canvas</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/canvas/(name)</td>
</tr>
<tr>
<td>Annotation</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/annotation/(name)</td>
</tr>
<tr>
<td>AnnotationList</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/list/(name)</td>
</tr>
<tr>
<td>Range</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/range/(name)</td>
</tr>
<tr>
<td>Layer</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/layer/(name)</td>
</tr>
<tr>
<td>Content</td>
<td>{scheme}://(host)&gt;({prefix}/identifier/res/((name), (format)</td>
</tr>
</tbody>
</table>

B. Summary of Metadata Requirements

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>☒</td>
<td>Required</td>
</tr>
<tr>
<td>☑</td>
<td>Recommended</td>
</tr>
<tr>
<td>○</td>
<td>Optional</td>
</tr>
<tr>
<td>✗</td>
<td>Not Allowed</td>
</tr>
</tbody>
</table>

Descriptive and Rights Properties

<table>
<thead>
<tr>
<th>Field</th>
<th>label</th>
<th>metadata</th>
<th>description</th>
<th>thumbnail</th>
<th>attribution</th>
<th>license</th>
<th>logo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manifest</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canvas</td>
<td>☒</td>
<td></td>
<td>☑</td>
<td></td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Annotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AnnotationList</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>label</td>
<td>metadata</td>
<td>description</td>
<td>thumbnail</td>
<td>attribution</td>
<td>license</td>
<td>logo</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>Range</td>
<td>🟠</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Layer</td>
<td>🟠</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Content</td>
<td>🟠</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Content</td>
<td>🟠</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Technical Properties**

<table>
<thead>
<tr>
<th></th>
<th>@id</th>
<th>@type</th>
<th>format</th>
<th>height</th>
<th>width</th>
<th>viewingDirection</th>
<th>viewingHint</th>
<th>navDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Manifest</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Sequence</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Canvas</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Annotation</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>AnnotationList</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Range</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Layer</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Image Content</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Other Content</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
</tbody>
</table>

**Linking Properties**

<table>
<thead>
<tr>
<th></th>
<th>seeAlso</th>
<th>service</th>
<th>related</th>
<th>rendering</th>
<th>within</th>
<th>startCanvas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Manifest</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Sequence</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Canvas</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Annotation</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>AnnotationList</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Range</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Layer</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Image Content</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Other Content</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
</tbody>
</table>

**Paging Properties**

<table>
<thead>
<tr>
<th></th>
<th>first</th>
<th>last</th>
<th>total</th>
<th>next</th>
<th>prev</th>
<th>startIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Manifest</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
<tr>
<td>Sequence</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
<td>🟠</td>
</tr>
</tbody>
</table>
C. Example Manifest Response

URL: http://example.org/iiif/book1/manifest
D. Implementation Notes

- Clients SHOULD be aware that some implementations may add an @graph property at the top level, which contains the object. This is a side effect of JSON-LD (JSON for Linking Data) serialization, and servers SHOULD remove it before sending to the client. If this is seen in practice, the client can use the JSON-LD (JSON for Linking Data) compaction algorithm (http://www.w3.org/TR/json-ld-api/#compaction-algorithms) and JSON-LD (JSON for Linking Data) Framing with the supplied frames (/api/annex/notes/jsonld/) to remove it and generate the correct representation.

- If a {name} parameter in the recommended URI structure begins with a number, such as .../canvas/1, then developers using certain technology stacks may be inconvenienced. In particular, an RDF (Resource Description Framework) based stack that uses RDF (Resource Description Framework)/XML (eXtensible Markup Language) internally will not be able to derive a shared .../canvas/ prefix and then use the 1 as a CURIE, as <canvas:1> is not a valid element in XML (eXtensible Markup Language). Producers might consider adding an alphabetical character as the initial character.

E. Versioning

Starting with version 2.0, this specification follows Semantic Versioning (http://semver.org/spec/v2.0.0.html). See the note Versioning of APIs (/api/annex/notes/semver/) for details regarding how this is implemented.

F. Acknowledgements

The production of this document was generously supported by a grant from the Andrew W. Mellon Foundation (http://www.mellon.org/).

Many thanks to the members of the IIIF (International Image Interoperability Framework) (https://iiif.io/community/) for their continuous engagement, innovative ideas and feedback.

G. Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-06-09</td>
<td>Version 2.1.1 View change log (/api/presentation/2.1/change-log-211/)</td>
</tr>
<tr>
<td>2016-05-12</td>
<td>Version 2.1 (Hinty McHintface) View change log (/api/presentation/2.1/change-log/)</td>
</tr>
<tr>
<td>2014-09-11</td>
<td>Version 2.0 (Triumphant Giraffe) View change log (/api/presentation/2.0/change-log/)</td>
</tr>
<tr>
<td>2013-08-26</td>
<td>Version 1.0 (unnamed)</td>
</tr>
<tr>
<td>2013-06-14</td>
<td>Version 0.9 (unnamed)</td>
</tr>
</tbody>
</table>

Feedback: iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com)  Get involved: Join IIIF (https://iiif.io/community/#how-to-get-involved)
IIIF Content Search API 1.0

Status of this Document

This Version: 1.0.0
Latest Stable Version: 1.0.0 (/api/search/1.0/)

Editors

Michael Appleby (https://orcid.org/0000-0002-1266-298X) (https://orcid.org/0000-0002-1266-298X), Yale University (http://www.yale.edu)

Tom Crane (https://orcid.org/0000-0003-1881-243X) (https://orcid.org/0000-0003-1881-243X), Digirati (http://digirati.com)

Robert Sanderson (https://orcid.org/0000-0003-4441-6852) (https://orcid.org/0000-0003-4441-6852), Stanford University (http://www.stanford.edu)

Jon Stroop (https://orcid.org/0000-0002-0367-1243) (https://orcid.org/0000-0002-0367-1243), Princeton University Library (https://library.princeton.edu)

Simeon Warner (https://orcid.org/0000-0002-7970-7855) (https://orcid.org/0000-0002-7970-7855), Cornell University (https://www.cornell.edu)

Copyright © 2015-2019 Editors and contributors. Published by the IIIF (International Image Interoperability Framework) Consortium under the CC-BY (http://creativecommons.org/licenses/by/4.0/) license, see disclaimer (/api/annex/notes/disclaimer/).

Table of Contents

1. Introduction
   1.1. Use Cases
   1.2. Terminology

2. Overview

3. Search
   3.1. Service Description
1. Introduction

In the IIIF (International Image Interoperability Framework) (pronounced “Triple-Eye-Eff”) Presentation API (Application Programming Interface) (/api/presentation/2.1/), content is brought together from distributed systems via annotations. That content might include images, often with a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service to access them, audio, video, rich or plain text, or anything else. In a vibrant and dynamic system, that content can come from many sources and be rich, varied and abundant. Of that list of content types, textual resources lend themselves to being searched, either as the transcription, translation or edition of the intellectual content, or commentary, description, tagging or other annotations about the object.

This specification lays out the interoperability mechanism for performing these searches within the IIIF (International Image Interoperability Framework) context. The scope of the specification is searching annotation content within a single IIIF (International Image Interoperability Framework) resource, such as a Manifest, Range or Collection. Every effort is made to keep the interaction as consistent with existing IIIF (International Image Interoperability Framework) patterns as possible. Searching for metadata or other descriptive properties is not in scope for this work.

In order to make searches easier against unknown content, a related service for the auto-completion of search terms is also specified. The auto-complete service is specific to a search service to ensure that the retrieved terms can simply be copied to the query of the search.

Please send feedback to iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com)

1.1. Use Cases
Use cases for being able to search the annotations within the Presentation API (Application Programming Interface) include:

- Searching OCR generated text to find words or phrases within a book, newspaper or other primarily textual content.
- Searching transcribed content, provided by crowd-sourcing or transformation of scholarly output.
- Searching multiple streams of content, such as the translation or edition, rather than the raw transcription of the content, to jump to the appropriate part of an object.
- Searching on sections of text, such as defined chapters or articles.
- Searching for user provided commentary about the resource, either as a discovery mechanism for the resource or for the discussion.
- Discovering similar sections of text to compare either the content or the object.

User interfaces that could be built using the search response include highlighting matching words in the display, providing a heatmap of where the matches occur within the object, and providing a mechanism to jump between points within the object. The auto-complete service assists users in identifying terms that exist within the selected scope.

1.2. Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC (Request for Comments) 2119 (http://tools.ietf.org/html/rfc2119).

2. Overview

The IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) (/api/presentation/2.1/) provides just enough information to a viewer so that it can present the images and other content to the user in a rich and understandable way. Those content resources may have textual annotations associated with them. Annotations may also be associated with the structural components of the Presentation API (Application Programming Interface), such as the manifest itself, sequences, ranges, and layers. Further, annotations can be replied to by annotating them to form a threaded discussion about the commentary, transcription, edition or translation.

Annotations are typically made available to viewing applications in an annotation list, where all of the annotations in the list target the same resource, or part of it. Where known, these lists can be directly referenced from the manifest document to allow clients to simply follow the link to retrieve them. For fixed, curated content, this is an appropriate method to discover them, as the annotations do not frequently change, nor are they potentially distributed amongst multiple servers. Annotation lists can be included in layers to group them together, such as by the source of the annotations, to allow the user to manipulate that grouping as a whole.

However this is less useful for comment-style annotations, crowd-sourced or distributed transcriptions, corrections to automated OCR transcription, and similar, as the annotations may be in constant flux. Further, being able to quickly discover individual annotations without stepping through all of the views of an object is essential for a reasonable user experience. This specification adds this capability to the IIIF (International Image Interoperability Framework) suite of specifications.

Beyond the ability to search for words or phrases, users find it helpful to have suggestions for what terms they should be searching for. This facility is often called auto-complete or type-ahead, and within the context of a single object can provide insight into the language and content. The auto-complete service is associated with a search service into which the terms can be fed as part of a query.

3. Search
The search service takes a query, including typically a search term or URI, and potentially filtering further by other properties including the date the annotation was created or last modified, the motivation for the annotation, or the user that created the annotation.

3.1. Service Description

Any resource in the Presentation API (Application Programming Interface) may have a search service associated with it. The resource determines the scope of the content that will be searched. A service associated with a manifest will search all of the annotations on canvases or other objects below the manifest, a service associated with a particular range will only search the canvases within the range, or a service on a canvas will search only annotations on that particular canvas.

The description of the service follows the pattern established in the Linking to Services (/api/annex/services/) specification. The description block MUST have the @context property with the value “http://iiif.io/api/search/1/context.json”, the profile property with the value “http://iiif.io/api/search/1/search”, and the @id property that contains the URI where the search can be performed.

An example service description block:

```json
{
    // ... the resource that the search service is associated with ...
    "service": {
      "@context": "http://iiif.io/api/search/1/context.json",
      "@id": "http://example.org/services/identifier/search",
      "profile": "http://iiif.io/api/search/1/search"
    }
}
```

3.2. Request

The search request is made to a service that is related to a particular Presentation API (Application Programming Interface) resource. The URIs for services associated with different resources must be different to allow the client to use the correct one for the desired scope of the search. To perform a search, the client MUST use HTTP (Hypertext Transfer Protocol) GET (rather than POST) to make the request to the service, with query parameters to specify the search terms.

3.2.1. Query Parameters

Other than q, which is RECOMMENDED, all other parameters are OPTIONAL in the request. The default, if a parameter is empty or not supplied, is to not restrict the annotations that match the search by that parameter. If the value is supplied but the field is not present in an annotation, then the search does not match that annotation. For example if an annotation does not have a creator, and the query specifies a user parameter, then the annotation does not match the query.

Servers SHOULD implement the q and motivation parameters and MAY implement the other parameters. Parameters that are received in a request but not implemented MUST be ignored, and SHOULD be included in the ignored property of the Layer in the response, described below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>A space separated list of search terms. The search terms MAY be either words (to search for within textual bodies) or URIs (to search identities of annotation body resources). The semantics of multiple, space separated terms is server implementation dependent.</td>
</tr>
<tr>
<td>motivation</td>
<td>A space separated list of motivation terms. If multiple motivations are supplied, an annotation matches the search if any of the motivations are present. Expected values are given below.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>date</td>
<td>A space separated list of date ranges. An annotation matches if the date on which it was created falls within any of the supplied date ranges. The dates MUST be supplied in the ISO8601 format: YYYY-MM-DDThh:mm:ssZ/YYYY-MM-DDThh:mm:ssZ. The dates MUST be expressed in UTC and MUST be given in the Z based format.</td>
</tr>
<tr>
<td>user</td>
<td>A space separated list of URIs that are the identities of users. If multiple users are supplied, an annotation matches the search if any of the users created the annotation.</td>
</tr>
</tbody>
</table>

Common values for the motivation parameter are:

<table>
<thead>
<tr>
<th>Motivation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>painting</td>
<td>Only annotations with the sc:painting motivation</td>
</tr>
<tr>
<td>non-painting</td>
<td>Annotations with any motivation other than sc:painting</td>
</tr>
<tr>
<td>commenting</td>
<td>Annotations with the oa:commenting motivation</td>
</tr>
<tr>
<td>describing</td>
<td>Annotations with the oa:describing motivation</td>
</tr>
<tr>
<td>tagging</td>
<td>Annotations with the oa:tagging motivation</td>
</tr>
<tr>
<td>linking</td>
<td>Annotations with the oa:linking motivation</td>
</tr>
</tbody>
</table>

Other motivations are possible, and the full list from the Open Annotation (http://www.openannotation.org/spec/core/) specification SHOULD be available by dropping the “oa:” prefix. Other, community specific motivations SHOULD include a prefix or use their full URI.

3.2.2. Example Request

This example request:

```
http://example.org/services/manifest/search?q=bird&motivation=painting
```

Would search for annotations with the word “bird” in their textual content, and have the motivation of painting. It would search annotations within the resource the service was associated with.

3.3. Presentation API (Application Programming Interface) Compatible Responses

The response from the server is an annotation list (/api/presentation/2.1/#annotation-list), following the format from the Presentation API (Application Programming Interface) with a few additional features. This allows clients that already implement the AnnotationList format to avoid further implementation work to support search results.

The search results are returned as annotations in the regular IIIF (International Image Interoperability Framework) syntax. Note that the annotations can come from multiple canvases, rather than the default situation from the Presentation API (Application Programming Interface) where all of the annotations target a single canvas.

3.3.1. Simple Lists

The simplest response looks exactly like a regular annotation list, where all of the matching annotations are returned in a single response. The value of @id will be the same as the URI used in the query, however servers MAY drop query parameters that are ignored so long as they are reported in the ignored property.
Clients wishing to know the total number of annotations that match may count the number of annotations in the resources property, as all matches have been returned. The full annotation description MUST be included in the response, even if the annotations are separately dereferenceable via their URIs.

```json
{
    "@context": "http://iiif.io/api/presentation/2/context.json",
    "@id": "http://example.org/service/manifest/search?q=bird&motivation=painting",
    "@type": "sc:AnnotationList",

    "resources": [
        {
            "@id": "http://example.org/identifier/annotation/anno-line",
            "@type": "oa:Annotation",
            "motivation": "sc:painting",
            "resource": {
                "@type": "cnt:ContentAsText",
                "chars": "A bird in the hand is worth two in the bush"
            },
            "on": "http://example.org/identifier/canvas1#xywh=100,100,250,20"
        }
    ]
}
```

3.3.2. Paging Results

For long lists of annotations, the server may choose to divide the response into multiple sections, often called pages. Each page is an annotation list and can refer to other pages to allow the client to traverse the entire set. This uses the paging features (/api/presentation/2.1/) introduced in version 2.1 of the Presentation API (Application Programming Interface), but is backwards compatible with version 2.0. The next page of results that follows the current response MUST be referenced in a `next` property of the annotation list, and the previous page SHOULD be referenced in a `prev` property.

The URI of the first annotation list reported in the `@id` property MAY be different from the one used by the client to request the search. Each page SHOULD also have a `startIndex` property with an integer value that reports the position of the first result within the entire result set, where the first annotation has an index of 0. For example, if the client has requested the first page which has 10 hits, then the `startIndex` will be 0, and the `startIndex` of second page will be 10, being the 11th hit.

All of the pages are within a layer (/api/presentation/2.1/#layer) that represents the entire resultset of matched annotations. The layer is the value of a `within` property on each of the page annotation lists, and is recorded as an object with properties.

The layer MUST have the `@type` property, with the value of "sc:Layer". It SHOULD refer to the URIs of the first and last annotation list pages with `first` and `last` properties, respectively. The layer SHOULD have a `total` property which is the total number of hits generated by the query, and it MAY have a URI given as the value of the `@id` property.

An example request:

```
http://example.org/service/manifest/search?q=bird
```

And the response for the first page of annotations from a total of 125 matches:
3.3.3. Target Resource Structure

The annotations may also include references to the structure or structures that the target (the resource in the on property) is found within. The URI and type of the including resource MUST be given, and a label SHOULD be included.

This structure is called out explicitly as although it uses only properties from the Presentation API (Application Programming Interface), it is not a common pattern and thus clients may not be expecting it.
3.4 Search API (Application Programming Interface) Specific Responses

There may be properties that are specific to the search result, and not features of the annotation in general, that are valuable to return to the client. Examples of such properties include the text before and after the matched content (to allow a result snippet to be presented), the matched text itself (when case normalization, stemming or wildcards have been applied), and a reference to the set of annotations that together fulfill the search query (when a phrase spans across multiple annotations).

As these responses include Search specific information, the value of @context MUST be an array with both the Presentation API (Application Programming Interface) and the Search API (Application Programming Interface) context URIs included, in that order. This allows the two APIs to develop separately and yet remain as synchronized as possible.

To incrementally build upon existing solutions and provide graceful degradation for clients that do not support these features and retain compatibility with the Presentation API (Application Programming Interface), the search API (Application Programming Interface) specific information is included in a second list within the annotation list called hits, other than the ignored property on the layer. Annotation lists MAY have this property, and servers MAY support these features.

If supported, each entry in the hits list is a search:Hit object. This type must be included as the value of the @type property. Hit objects reference one or more annotations that they provide additional information for, in a list as the value of the hit’s annotations property. The reference is made to the value of the @id property of the annotation, and thus annotations MUST have a URI to enable this further information.

The basic structure is:
3.4.1. Ignored Parameters

If the server has ignored any of the parameters in the request, then the layer MUST be present and MUST have an ignored property where the value is a list of the ignored parameters.

If the request from previous examples had been:

```
http://example.org/service/manifest/search?q=bird&user=http%3A%2F%2Fexample.com%2Fusers%2Fazaroth42
```

And the user parameter was ignored when processing the request, the response would be:
3.4.2. Search Term Snippets

The simplest addition to the hit object is to add text that appears before and after the matching text in the annotation. This allows the client to construct a snippet where the matching text is provided in the context of surrounding content, rather than simply by itself. This is most useful when the service has word-level boundaries of the text on the canvas, such as are available when Optical Character Recognition (OCR) has been used to generate the text positions.

The service MAY add a before property to the hit with some amount of text that appears before the content of the annotation (given in chars), and MAY also add an after property with some amount of text that appears after the content of the annotation.

For example, in a search for the query term “bird” in our example sentence, when the server has full word level coordinates:

```
http://example.org/service/manifest/search?q=bird
```

That the server matches against the plural “birds”:
3.4.3. Search Term Highlighting

Many systems do not have full word-level coordinate information, and are restricted to line or paragraph level boundaries. In this case the most useful thing that the client can do is to display the entire annotation and highlight the hits within it. This is similar, but different, to the previous use case. Here the word will appear somewhere within the chars property of the annotation, and the client needs to make it more prominent. In the previous situation, the word was the entire content of the annotation, and the information was convenient for presenting it in a list.

The client in this case needs to know the text that caused the service to create the hit, and enough information about where it occurs in the content to reliably highlight it and not highlight non-matches. To do this, the service can supply text before and after the matching term within the content of the annotation, via an Open Annotation (http://www.openannotation.org/spec/core/) TextQuoteSelector object. TextQuoteSelectors have three properties: exact to record the exact text to look for, prefix with some text before the match, and suffix with some text after the match.

This would look like:
As multiple words might match the query within the same annotation, multiple selectors may be given in the hit as objects within a selectors property. For example, if the search used a wildcard to search for all words starting with “b” it would match the same annotation twice:

```
http://example.org/service/manifest/search?q=b*
```

The result might be:
3.4.4. Multi-Annotations Hits

Given the flexibility of alignment between the sections of the text (such as word, line, paragraph, page, or arbitrary sections) and the annotations that expose that text to the client, there may be multiple annotations that match a single multi-term search. These differences will depend primarily on the method by which the text and annotations were generated and will likely be very different for manually transcribed texts and text that it is generated by OCR.
For example, imagine that the annotations are divided up line by line, as they were manually transcribed that way, and there are two lines of text. In this example the first line is “A bird in the hand”, the second line is “is worth two in the bush”, and the search is for the phrase “hand is”. Therefore the match spans both of the line-based annotations. If the annotations were instead at word level, then all phrase searches would require multiple annotations.

In cases like this there are more annotations than hits as two or more annotations are needed to make up one hit. The match property of the hit captures the text across the annotations.

```json
{
    "@context": [
        "http://iiif.io/api/presentation/2/context.json",
        "http://iiif.io/api/search/1/context.json"
    ],
    "@id": "http://example.org/service/manifest/search?q=hand+is",
    "@type": "sc:AnnotationList",
    "resources": [
        {
            "@id": "http://example.org/identifier/annotation/anno-bird",
            "@type": "oa:Annotation",
            "motivation": "sc:painting",
            "resource": {
                "@type": "cnt:ContentAsText",
                "chars": "A bird in the hand"
            },
            "on": "http://example.org/identifier/canvas1#xywh=200,100,150,30"
        },
        {
            "@id": "http://example.org/identifier/annotation/anno-are",
            "@type": "oa:Annotation",
            "motivation": "sc:painting",
            "resource": {
                "@type": "cnt:ContentAsText",
                "chars": "is worth two in the bush"
            },
            "on": "http://example.org/identifier/canvas1#xywh=200,140,170,30"
        }
        // Further annotations here ...
    ],
    "hits": [
        {
            "@type": "search:Hit",
            "annotations": [
                "http://example.org/identifier/annotation/anno-bush",
                "http://example.org/identifier/annotation/anno-are"
            ],
            "match": "hand is",
            "before": "A bird in the",
            "after": " worth two in the bush"
        }
        // Further hits for the first page here ...
    ]
}
```
4. Autocomplete

The autocomplete service returns terms that can be added into the q parameter of the related search service, given the first characters of the term.

4.1. Service Description

The autocomplete service is nested within the search service that it provides term completion for. This is to allow multiple search services, each with their own autocomplete service.

The autocomplete service MUST have an @id property with the value of the URI where the service can be interacted with, and MUST have a profile property with the value “http://iiif.io/api/search/1/autocomplete" to distinguish it from other types of service.

```json
{
    "service": {
        "@context": "http://iiif.io/api/search/1/context.json",
        "@id": "http://example.org/services/identifier/search",
        "profile": "http://iiif.io/api/search/1/search",
        "service": {
            "@id": "http://example.org/services/identifier/autocomplete",
            "profile": "http://iiif.io/api/search/1/autocomplete"
        }
    }
}
```

4.2. Request

The request is very similar to the search request, with one additional parameter to allow the number of occurrences of the term within the object to be constrained. The value of the q parameter, which is REQUIRED for the autocomplete service, is the beginning characters from the term to be completed by the service. For example, the query term of ‘bir’ might complete to ‘bird’, ‘biro’, ‘birth’, and ‘birthday’.

The term should be parsed as a complete string, regardless of whether there is whitespace included in it. For example, the query term of “green bir” should not autocomplete on fields that match “green” and also include something that starts with “bir”, but instead look for terms that start with the string “green bir”.

The other parameters (motivation, date and user), if supported, refine the set of terms in the response to only ones from the annotations that match those filters. For example, if the motivation is given as “painting”, then only text from painting transcriptions will contribute to the list of terms in the response.

4.2.1. Query Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>The minimum number of occurrences for a term in the index in order for it to appear within the response ; default is 1 if not present. Support for this parameter is OPTIONAL</td>
</tr>
</tbody>
</table>

4.2.2. Example Request

An example request

```url
http://example.org/service/identifier/autocomplete?
q=bir&motivation=painting&user=http%3A%2F%2Fexample.com%2Fusers%2Fazaroth42
```

4.3. Response
The response is a list (a “search:TermList”) of simple objects that include the term, a link to the search for that term, and the number of matches that search will have. The number of terms provided in the list is determined by the server.

Parameters that were not processed by the service MUST be returned in the ignored property of the main “TermList” object. The value MUST be an array of strings.

The objects in the list of terms are all of @type “search:Term”, and this MAY be included explicitly but is not necessary. The Term object has a number of possible properties:

- The matching term is given as the value of the match property, and MUST be present.
- The link to the search to perform is the value of the url property, and this MUST be present.
- The number of matches for the term is the integer value of the count property, and SHOULD be present.
- A label to display instead of the match can be given as the value of the label property, and MAY be present. There may be more than one label given, to allow for internationalization.

The terms SHOULD be provided in ascending alphabetically sorted order, but other orders are allowed, such as by the term’s count descending to put the most common matches first.

The example request above might generate the following response:

```
{
  "@context": "http://iiif.io/api/search/1/context.json",
  "@id": "http://example.org/service/identifier/autocomplete?q=bir&motivation=painting",
  "@type": "search:TermList",
  "ignored": ["user"],
  "terms": [
    {
      "match": "bird",
      "url": "http://example.org/service/identifier/search?motivation=painting&q=bird",
      "count": 15
    },
    {
      "match": "biro",
      "url": "http://example.org/service/identifier/search?motivation=painting&q=biro",
      "count": 3
    },
    {
      "match": "birth",
      "url": "http://example.org/service/identifier/search?motivation=painting&q=birth",
      "count": 9
    },
    {
      "match": "birthday",
      "url": "http://example.org/service/identifier/search?motivation=painting&q=birthday",
      "count": 21
    }
  ]
}
```

It is also possible to associate one or more labels to display to the user with URIs or other data that are searchable via the q parameter, rather than using the exact string that matched. This can also be useful if stemming or other term normalization has occurred, in order to display the original rather than the processed term.
5. Property Definitions

**after**

The segment of text that occurs after the text that triggered the search to match the particular annotation. The value MUST be a single string.

- A Hit MAY have the after property.

**before**

The segment of text that occurs before the text that triggered the search to match the particular annotation. The value MUST be a single string.

- A Hit MAY have the before property.

**count**

The number of times that the term appears. The value MUST be an positive integer.

- A Term SHOULD have the count property.

**ignored**

The set of parameters that were received by the server but not taken into account when processing the query. The value MUST be an array of strings.

- A TermList or a Layer MAY have an ignored property, and MUST have it if the server ignored any query parameter.

**match**

The text that triggered the search to match the particular annotation. The value MUST be a single string.

- A Hit MAY have the match property.
- A Term MUST have the match property.

Appendices

A. Request Parameter Requirements
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required in Request</th>
<th>Required in Search</th>
<th>Required in Autocomplete</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>recommended</td>
<td>recommended</td>
<td>mandatory</td>
</tr>
<tr>
<td>motivation</td>
<td>optional</td>
<td>recommended</td>
<td>optional</td>
</tr>
<tr>
<td>date</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>uri</td>
<td>optional</td>
<td>optional</td>
<td>optional</td>
</tr>
<tr>
<td>min</td>
<td>optional</td>
<td>n/a</td>
<td>optional</td>
</tr>
</tbody>
</table>

B. Versioning

This specification follows Semantic Versioning (http://semver.org/spec/v2.0.0.html). See the note Versioning of APIs (/api/annex/notes/semver/) for details regarding how this is implemented.

C. Acknowledgements

The production of this document was generously supported by a grant from the Andrew W. Mellon Foundation (http://www.mellon.org/).

Many thanks to the members of the IIIF (International Image Interoperability Framework) (https://iiif.io/community/) for their continuous engagement, innovative ideas and feedback.

D. Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-05-12</td>
<td>Version 1.0 (Lost Summer)</td>
</tr>
<tr>
<td>2015-07-20</td>
<td>Version 0.9 (Trip Life)</td>
</tr>
</tbody>
</table>

Feedback: iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com)  Get involved: Join IIIF (https://iiif.io/community/#how-to-get-involved)
Abstract

The SharedCanvas data model specifies a linked data based approach for describing digital facsimiles of physical objects in a collaborative fashion. It is intended for use in the cultural heritage domain, although may be useful in other areas, and is designed around requirements derived from digitized text-bearing objects such as medieval manuscripts. Instances of the data model are consumed by rendering platforms in order to understand the relationships between the constituent text, image, audio or other resources. These resources are associated with an abstract Canvas, or parts thereof, via Open Annotations and the Annotations are grouped and ordered in OAI-ORE Aggregations.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document.

This document has been made available to the International Image Interoperability Framework group and the Digital Medieval Manuscript group for review, but is not endorsed by them. This is a final working draft, with a last call for comments. It is inappropriate to refer to this document other than as "work in progress".

Please send general comments about this document to the public mailing list: http://groups.google.com/group/sharedcanvas.

Table of Contents

1. Introduction
   1.1. Namespaces
   1.2. Terminology
   1.3. Examples
2. Canvases
   2.1. Canvases
   2.2. Zones
3. Annotations
   3.1. Annotation Recommendations
1. Introduction

There are many digital repositories that maintain hundreds of thousands of page images of medieval manuscripts or other historically important, handwritten documents. These images act as surrogates for physical objects and are often the only way in which scholars and students can interact with the material, making it essential that the experience be rich with access to as much of the research about the physical objects as possible.

Repositories have two main tasks in this domain, digitization of the original and then providing access to the digital resources. While the former is a process specific to the institution, it was recognized that the latter would benefit greatly from a shared data model and implementations to facilitate the creation of software environments to consume that model. It was also recognized that the knowledge about the object is often maintained across multiple repositories, and any sustainable solution would need to embrace this aspect rather than continue with current "silo" based approaches, where a single repository holds all of the information.

The goal of the Shared Canvas data model is to provide a standardized description of the digital resources that are surrogates for culturally important, primarily textual physical objects in order to enable interoperability between repositories, tools, services and presentation systems. The resource being modeled is, therefore, a single physical item (a book, a manuscript, a painting) rather than the intellectual work embodied within it (the text or other content). The modeling requirements are drawn from presentation systems use cases only, and the data model does not attempt to include descriptive metadata about the intellectual work other than that which is necessary to display to the user as immediate context in a user interface.

Much of the effort to display this digitized material has been duplicated across institutions with each recreating very similar basic page-turning applications; with the SharedCanvas data model and jointly developed viewing environments, this duplicated effort may be repurposed to further digitization or better descriptive metadata. The approach also enables seamless interoperability between repositories who may hold related content, or even individual physical pages removed from their original context.

SharedCanvas adopts the principles of Linked Open Data and the Architecture of the World Wide Web, in recognition of the importance of this distributed information creation, management and ownership. The innovation described in this data model is a Linked Data Canvas, an abstract space that represents a page of the digitized item, which has resources painted on to it in a distributed manner via Annotations. The Canvas is given its own globally unique identifier, an HTTP URI, and thus additional information may be associated with the
Canvas by anybody. It is the task of the rendering environments to determine if that information is appropriate for display to the current audience.

The association is done by Annotation, where the annotated resource is the Canvas and the "comment" is the resource to be displayed. The Annotations may be stored anywhere, and are collected together into Aggregations for ease of consumption. It is the role of the repository to both create and curate these annotations to provide trustworthy sets of information to the rendering environment.

**1.1 Namespaces**

The Shared Canvas model defines a single namespace, which includes all of the new classes, relationships and properties defined.

The ontology for the namespace is intended to be static and will always use the same namespace URI. All versions of the ontology will remain available from version specific URLs, and the namespace URI will provide access to the most recent version.

The following namespaces are used in this document:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td><a href="http://www.shared-canvas.org/ns/">http://www.shared-canvas.org/ns/</a></td>
<td>Shared Canvas Ontology</td>
</tr>
<tr>
<td>cnt</td>
<td><a href="http://www.w3.org/2011/content#">http://www.w3.org/2011/content#</a></td>
<td>Representing Content in RDF</td>
</tr>
<tr>
<td>dc</td>
<td><a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a></td>
<td>Dublin Core Elements</td>
</tr>
<tr>
<td>dcterms</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
<td>Dublin Core Terms</td>
</tr>
<tr>
<td>dctypes</td>
<td><a href="http://purl.org/dc/dcmitype/">http://purl.org/dc/dcmitype/</a></td>
<td>Dublin Core types</td>
</tr>
<tr>
<td>exif</td>
<td><a href="http://www.w3.org/2003/12/exif/ns#">http://www.w3.org/2003/12/exif/ns#</a></td>
<td>Exif vocabulary</td>
</tr>
<tr>
<td>oa</td>
<td><a href="http://www.w3.org/ns/oa#">http://www.w3.org/ns/oa#</a></td>
<td>The Open Annotation model</td>
</tr>
<tr>
<td>ore</td>
<td><a href="http://www.openarchives.org/ore/terms/">http://www.openarchives.org/ore/terms/</a></td>
<td>The Open Archives Object Re-Use and Exchange vocabulary</td>
</tr>
<tr>
<td>rdf</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
<td>RDF Vocabulary</td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>RDF Schema Vocabulary</td>
</tr>
</tbody>
</table>

**1.2 Terminology**

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

**1.3 Examples**

Examples used throughout the document will be conveyed as both a diagram and in the Turtle RDF format, and do not represent specific use cases with real resources. The Turtle examples do not provide namespace declarations, and should be considered to follow the Namespaces table in Section 1.1. Usage examples in SPARQL are given for each section, based on a requirement expressed in natural language. Additional examples of how to model and implement specific situations are available in the Shared Canvas Cookbook.
The diagrams in the specification use the following style:

- Instances are depicted as colored ellipses
  - Instances with a resolvable URI have a single line border
  - Instances with a non-resolvable URN or are a blank node have a double line border
- Classes are depicted as white rectangles
- Literals are depicted as white lozenges
- Relationships are depicted as straight, black lines. Relationships are RDF predicates where the range is a Resource, and equivalent to OWL object properties.
- Properties are depicted as curved, black lines. Properties are RDF predicates where the range is a Literal, and equivalent to OWL datatype properties.
- Class instantiation (\texttt{rdf:type}) is depicted as a straight black line with white arrow head
- Example instance identifiers or example literal values always end in a number. For example, \texttt{canvas1} is a specific instance of an Canvas, whereas '\texttt{sc:Canvas}' is a class
- Relationships not explicit in the model, but important for understanding, are depicted as curved, dashed, colored lines
- Conceptual resource boundaries not explicit in the model, but considered important for understanding, are depicted as grey dashed boxes around the components. They are used to convey spatial parts of the diagrams and may be safely ignored.

2. Canvas Model

A Shared Canvas is a two dimensional rectangular space with an aspect ratio that represents a single logical view of some part of the physical item. For example, a book would have one Canvas per side of a page, a painting would have a single Canvas and a scroll might have either one very long Canvas representing the entire object, or multiple smaller ones if it made more sense to view it in sections.

Canvases are a crucial construction in the model, as they are the single point that can collect and layout multiple content resources that make up the facsimile. A single physical page may be depicted in multiple digitizations of various qualities, lighting conditions, size, format or other aspects, and the Canvas is the single point where these images are aligned. Equally, the image may contain more than just the page and thus part of the image should be aligned with the entire Canvas. Even if there is currently only one image, in the future there may be a second or further digitizations performed.

Canvases also allow pages to be modeled that have not been digitized, but about which information is known. This might include pages that are lost, destroyed or hypothetical, but have been transcribed either before the loss, or from another witness of the same text. It also allows books that have been partially digitized to be represented in their entirety for when the remaining images become available.

In the historical domain, there are many cases in which only fragments of the original page remain. These may be digitized and associated with the correct part of the Canvas that represents the original page. Multiple fragments are often digitized together, yet come from different original pages, and thus part of the image must be associated with part of a Canvas.

None of these use cases are able to be handled in a model in which Images rather than abstract Canvases are used for representing the aspect of the physical item.

2.1 Canvas
A Canvas is the digital surrogate for a physical page which should be rendered to the user. Each Canvas has a rectangular aspect ratio, and is positioned such that the top left hand corner of the Canvas corresponds to the top left hand corner of a rectangular bounding box around the page, and similarly for the bottom right hand corners. The identifier for the Canvas is not an identifier for the physical page, it identifies the digital representation of it.

![Canvas/Page Correspondence](image)

It is RECOMMENDED that the aspect ratio be expressed in such a way that alignment of images and other resources does not require floating point values to give a reasonable level of accuracy. Thus, values in the 1000 to 3000 range are more appropriate than in the 1 to 100 range.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Canvas</td>
<td>Class</td>
<td>The Class for a Canvas, which is the digital surrogate for a physical page within the model</td>
</tr>
<tr>
<td>exif:height</td>
<td>Property</td>
<td>The height of the Canvas, in no particular units. The height is given only to form part of the aspect ratio for the Canvas, along with the width. Each Canvas MUST have exactly one height.</td>
</tr>
<tr>
<td>exif:width</td>
<td>Property</td>
<td>The width of the Canvas, in no particular units. Along with height, this forms the aspect ratio of the Canvas. Each Canvas MUST have exactly one width.</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Property</td>
<td>The human readable label intended to be displayed to a user when viewing the Canvas. Each Canvas MUST have at least one label. Multiple labels MAY be expressed in different languages and the rendering system should select an appropriate one.</td>
</tr>
</tbody>
</table>
The relationship between a Canvas and a list of Annotations that target it or part of it. Each Canvas MAY have one or more lists of related annotations.

Canvas Model:

```
<Canvas1> a sc:Canvas ;
    exif:height 1400 ;
    exif:width 1000 ;
    rdfs:label "Page 1" .
```

2.2. Zones

A Zone is similar to a Canvas, as it is an abstract space with its own dimensions. Information may be associated with the Zone resource in order to manipulate all such resources as a single whole. This allows for the generation of dynamic user interface features such as manipulating folds in a piece of paper, or rotating a section of the page to be at the correct angle for ease of reading, without rotating the entire page. Zones may be associated with multiple Canvases, thus avoiding unnecessary repetition of the data.
Like Canvases, Zones have a height and width to provide an aspect ratio, and a label for choices and interpretation. Zones may also have a `sc:naturalAngle` property to define the rotation for easy of reading. They are associated with Canvases via Annotations, as described in the next section.

Please Note: Zones are an advanced modeling component, and while they are important for several important use cases, it should be expected that not every Shared Canvas implementation will support them.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Zone</td>
<td>Class</td>
<td>The class for Zones, which represent part of one or more Canvases</td>
</tr>
<tr>
<td>exif:height</td>
<td>Property</td>
<td>The height component of the aspect ratio of the Zone. Each Zone MUST have exactly 1 height given.</td>
</tr>
<tr>
<td>exif:width</td>
<td>Property</td>
<td>The width component of the aspect ratio of the Zone. Each Zone MUST have exactly 1 width given.</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Property</td>
<td>A human readable label for the Zone. Each Zone SHOULD have 1 or more labels</td>
</tr>
<tr>
<td>sc:naturalAngle</td>
<td>Property</td>
<td>The angle to which the Zone should be rotated to make the content easier to read for a user. Each sc:Zone MAY have exactly 1 sc:naturalAngle property and MUST NOT have more than 1.</td>
</tr>
</tbody>
</table>
3. Annotations

It is important to allow the distributed association of content resources, such as images and text, with the Canvas. This allows a separation of concerns where one institution might provide images, another the transcribed text, and a third scholarly commentary about the page being represented. Following previous systems, an annotation paradigm is used to model this association, not only for transcription but also for any content resource that should be rendered as part of the digital facsimile of the page.

Each content resource is, typically, used as the body of its own Annotation, and the target of the Annotation (the resource the Annotation is about) is the Canvas. Thus an image of a page may be associated with the Canvas as depicted below.
is crucial for the fragment use case, amongst others.

It is important to become familiar with the Open Annotation model, as this section provides additional information and recommendations based upon it.

Using Annotations for both commentary and painting the Canvas reduces the number of technologies used, making implementation of the model easier and more consistent. As Annotations are resources in their own right, not part of the Canvas directly, it enables a distributed model of creation and management of resources. The creator of the Annotation is not required to be the owner of the resource being annotated, or painted, on to the Canvas, nor the owner of the Canvas, enabling a crowd-sourced model and for users to easily contribute their knowledge.

Disagreement in the reconstruction of a resource can easily be modeled using Annotations, as differing opinions are represented by different Annotations with different annotators. It is then up to the client to determine which is correct, or present the various options to the user if that is more appropriate.

3.1. Painting Motivation

Shared Canvas defines a new instance of `oa:Motivation` called `sc:painting`. This motivation allows a client system to easily distinguish between Annotations that associate Content to be rendered as part of the facsimile and other Annotations, which might be used for scholarly commentary, moderation, discussion or many other motivations. These other Annotations should use appropriate motivations, as described in the Open Annotation specification or by further communities.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:painting</td>
<td>Instance</td>
<td>[instance of oa:Motivation] The motivation that represents the distinction between resources that should be painted onto the Canvas, rather than resources that are about the Canvas. If the target of the Annotation is not a Canvas or Zone, then the meaning is left to other communities to define.</td>
</tr>
</tbody>
</table>

**Annotation Model:**

![Annotation Model Diagram](http://www.shared-canvas.org/datamodel/spec/)
Please see the [Cookbook](http://www.shared-canvas.org/datamodel/spec/) for further usage of Annotations within the Shared Canvas model.

## 4. Ordering Model

A single Canvas can be used to render one side of a page, or perhaps an open scroll or newspaper, however multiple Canvases are required to create a page turning application in which a user can leaf through a book. The order in which these Canvases are displayed is, of course, crucial to ensuring a reasonable experience. In a graph model such as RDF order is not inherent in the serialization, such as is the case for XML, JSON or other data models, and thus the order must be explicitly described within the graph.

The Shared Canvas model starts from the [Object Reuse and Exchange specification](http://www.shared-canvas.org/datamodel/spec/), which provides a method for ordering based on Proxy nodes, however we introduce a simpler method for the most common case of a single, linear order.

### 4.1. Ordered Aggregations

Resources in RDF may be given multiple classes, thus at the same time a resource may be both an ore:Aggregation and an rdf:List. Using this technique, it is possible to express a single linear order for the aggregated resources without the expense of minting Proxy resources. The aggregated resources may be of any type, and the Shared Canvas specification uses it for both Canvases and other Aggregations.

Please note that all of the predicates associated with lists and aggregations are, of course, also used. This means that the Aggregation must have an associated Resource Map that has a creator and a modified timestamp. Not all of these relationships and properties are depicted in the model diagram below for reasons of space.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ore:Aggregation</td>
<td>Class</td>
<td>A set of aggregated resources, with some associated metadata about the set</td>
</tr>
<tr>
<td>rdf:List</td>
<td>Class</td>
<td>The class for ordering a set of resources, but without additional metadata</td>
</tr>
</tbody>
</table>

**Ordered Aggregation Model:**

```xml
<anno1> a oa:Annotation ;
oa:motivatedBy sc:painting ;
oa:hasBody <content1> ;
oa:hasTarget <canvas1> .

<canvas1> a sc:Canvas ;
exif:height 1400 ;
exif:width 1000 ;
rdfs:label "Page 1" .

<image1> a dctypes:Image ;
dc:format "image/jpeg" .
```
4.2. Sequences

A Sequence is the ordered aggregation of Canvases that should be used by the client to display the Canvases in a particular order to the user. There may be multiple Sequences with different orders for the same or overlapping set of Canvases. This is because a single set of pages may be rebound in different orders over time, or to have Sequences with only the pages that currently exist rather than the full set of original pages, some of which are now missing or destroyed.

The Sequence must express that it is an rdf:List for clients and systems that do not recognize the Shared Canvas specific class.

Model

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Sequence</td>
<td>Class</td>
<td>[subClass of ore:Aggregation] An ordered aggregation of Canvases for the purpose of rendering them in that order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A pointer to an sc:Range which contains</td>
</tr>
</tbody>
</table>
### 3.31.2014 Shared Canvas Data Model

<table>
<thead>
<tr>
<th>sc:hasContentRange</th>
<th>Relationship</th>
<th>the content bearing pages of the sequence. If sc:hasContentRange is not supplied, then it defaults to the entire Sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>Property</td>
<td>A human readable label for the Sequence. Each Sequence SHOULD have 1 or more labels</td>
</tr>
<tr>
<td>sc:readingDirection</td>
<td>Property</td>
<td>&quot;Left-to-Right&quot; or &quot;Right-to-Left&quot; for the reading direction of this sequence for animating page viewers.</td>
</tr>
</tbody>
</table>

**Sequence Model:**

![Diagram of Sequence Model](image)

```xml
<Seq1> sc:Sequence, rdf:List ;
   ore:aggregates <Canvas1>, <Canvas2>, <Canvas3> ;
   rdf:first <Canvas1> ;
   rdf:rest ( <Canvas2> <Canvas3> ) .

<ResourceMap1> a ore:ResourceMap ;
   ore:describes <Seq1> ;

<Canvas1> a sc:Canvas .
<Canvas2> a sc:Canvas .
<Canvas3> a sc:Canvas .
```

### 4.3. Ranges

The purpose of a Range is to provide a mechanism to select the Canvases and parts of Canvases that make up some intellectual entity. This might be based on the content, such as a story in an anthology, a verse of a poem or a chapter of a book, but equally may be used for any other purpose, such as identifying the flyleaves at the beginning or the non-contiguous set of pages that have illuminations or marginalia.

The basic structure of a Range is the same as a Sequence, but it may also aggregate segments of Canvases as well as entire Canvases. The segment is an oa:SpecificResource, with a source of the Canvas and a Selector that describes it. Please note that the Specific
Resource is likely to have a UUID URN as its identifier, and thus should NOT be part of an ORE Aggregation, which states that the aggregated resources MUST have "protocol-based" URIs. We knowingly ignore this requirement, as the Specific Resource is a placeholder resource for the combination of the protocol-based Source (admittedly an abstract Canvas), and the selector which describes the desired part of that Canvas.

The Range should also link to the Sequence or Sequences which constitute the full set of Canvases, with the dcterms:isPartOf relationship.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Range</td>
<td>Class</td>
<td>[subClass of ore:Aggregation] An ordered aggregation of Canvases for the purpose of rendering them in that order</td>
</tr>
<tr>
<td>dcterms:isPartOf</td>
<td>Relationship</td>
<td>The relationship between the Range and the Sequences which express the complete set of Canvases.</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Property</td>
<td>A human readable label for the Range. Each Range SHOULD have 1 or more labels</td>
</tr>
</tbody>
</table>

**Range Model:**

```
<Range1> a sc:Range, rdf:List ;
  dcterms:isPartOf <Seq1> ;
  ore:aggregates <Part1>, <Canvas2>, <Canvas3> ;
  rdf:first <Part1> ;
```

*Figure 4.3. Range Model*
5. Discovery Model

In an ideal world, client applications would have access to all of the RDF triples describing the entire dataset. Unfortunately this is not feasible, as there may be many millions of triples distributed around the web relating content to the Canvases, providing comments or additional metadata. Thus it is necessary to provide lists of pointers to the Annotations, Sequences and Ranges that are required to build up the presentation.

5.1. Annotation Lists

An Annotation List is another use of the Ordered Aggregation construction, but instead of aggregating Canvases, it aggregates Annotations. The Annotations needed for a particular resource may be divided up amongst any number of Lists using any criteria for determining which list or lists they should be part of. Typical divisions are per Canvas or based on the content type of the Body resource, such that all of the Annotations that associate an Image with a Canvas would be together.

Model

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:AnnotationList</td>
<td>Class</td>
<td>An ordered aggregation of Annotations</td>
</tr>
<tr>
<td>sc:forCanvas</td>
<td>Relationship</td>
<td>The relationship between the AnnotationList and the Canvas(es) which are the targets of the included Annotations. Typically this relationship is used to describe the AnnotationList in a Manifest to allow clients to determine which lists should be retrieved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A shortcut relationship that implies that all of the</td>
</tr>
</tbody>
</table>
Annotation List Model:

```xml
<AnnoList1> a sc:AnnotationList, ore:Aggregation, rdf:List ;
  ore:aggregates <Anno1>, <Anno2> ;
  rdf:first <Anno1> ;
  rdf:rest ( <Anno2> ) .

<ResourceMap1> a ore:ResourceMap ;
  ore:describes <AnnoList1> ;
  dcterms:modified "2011-08-24T19:09:59Z";
  dcterms:creator <Agent1>.

<Agent1> a dcterms:Agent ;
  foaf:name "Creator Name" .

<Anno1> a sc:ContentAnnotation .
<Anno2> a sc:ContentAnnotation .
```

Usage

Requirement: Find the Lists which include a particular Annotation:

```
SELECT ?aggr WHERE { ?aggr a sc:AnnotationList ; ?aggr ore:aggregates <Anno1> }
=> <AnnoList1>
```

5.2. Layers

It might be important to group together multiple AnnotationLists to describe the common features, perhaps along with some additional Annotations. For example, a Layer could be used to group AnnotationLists of Text Annotations based on a common feature. This would
enable the transcription annotations to be kept separate from edition annotations or translation annotations.

Layers are another instance of an Ordered Aggregation that may aggregate either Annotations or AnnotationLists, and MUST have an rdfs:label describing the purpose. All of the properties of AnnotationLists are also useful for Layers.

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Layer</td>
<td>Class</td>
<td>An ordered aggregation of Annotations or Annotation Lists.</td>
</tr>
</tbody>
</table>

5.3. Manifests

The Manifest is what ties everything together. It is an Aggregation of the Layers, AnnotationLists and Sequences that make up the description of the facsimile. As such the Manifest is representative of the Book, Newspaper, Scroll or whatever physical object is being represented in the facsimile.

A Manifest MUST have an rdf:label giving a human readable name for it. This label is to be used for rendering purposes to inform the user what they are looking at.

Model

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:Manifest</td>
<td>Class</td>
<td>An ordered aggregation of Annotations</td>
</tr>
<tr>
<td>sc:forCanvas</td>
<td>Relationship</td>
<td>The relationship between the AnnotationList and the Canvases which are the targets of the included Annotations. Typically this relationship is used to describe the AnnotationList in a Manifest to allow clients to determine which lists should be retrieved.</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Property</td>
<td>A human readable label for the Manifest. Each Manifest MUST have 1 or more labels</td>
</tr>
</tbody>
</table>

Annotation List Model:
Figure 5.2. Manifest Model

```xml
<Manifest1> a sc:Manifest, ore:Aggregation ;
  ore:aggregates <Seq1>, <Range1>, <AnnoList1> .

<ResourceMap1> a ore:ResourceMap ;
  ore:describes <Manifest1> ;
  dcterms:modified "2011-08-24T19:09:59Z";
  dcterms:creator <Agent1>.

<Agent1> a dcterms:Agent ;
  foaf:name "Creator Name" .

<Seq1> a sc:Sequence .
<Range1> a sc:Range .
<AnnoList1> a sc:AnnotationList ;
  sc:forCanvas <Canvas1> .
```

5.4. Collections

Shared Canvas does not address collections of Manifests (or of other Collections) directly. Ordered or regular ore:Aggregations are recommended as the basis of describing collections in a manner that would be compliant with the Shared Canvas guidelines.

5.5. Services and Bibliographic Information

The Shared Canvas data model does not directly address search services, or full bibliographic descriptions of the objects that are being rendered by the digital facsimiles. However, it does provide hooks for pointing to related services and descriptions and recommendations for appropriate APIs. These services should then either self-describe or be described in the RDF as to what type they are.
However, a few aspects are universally useful for rendering applications to have access to. The Manifest (as representative of the object as a whole), each Sequence and Canvas may have agents, dates and locations associated with them. These are not the creator or dates for the Manifest, Sequence, Range or Canvas itself but instead for the object or part thereof which they are depicting. Thus, as the Manifest may (should!) have its own creator, we need to be careful to distinguish which we are talking about. Thus, we create three "denormalized" fields in the Shared Canvas namespace that contain labels intended only for display to the end user, and is likely not to be suitable for search or any other purpose. More extensive bibliographic information should be referenced with the sc:hasRelatedDescription relationship.

**Model**

The Shared Canvas model uses:

<table>
<thead>
<tr>
<th>Vocabulary Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc:hasRelatedService</td>
<td>Relationship</td>
<td>The relationship between a resource in the Shared Canvas model and the endpoint for a related service.</td>
</tr>
<tr>
<td>sc:hasRelatedDescription</td>
<td>Relationship</td>
<td>The relationship between a resource in the Shared Canvas model and a related description of the real world object.</td>
</tr>
<tr>
<td>sc:agentLabel</td>
<td>Property</td>
<td>A name and possibly role of a person or organization associated with the physical object which is being represented by the Shared Canvas object. For example: &quot;Froissart (author)&quot;</td>
</tr>
<tr>
<td>sc:dateLabel</td>
<td>Property</td>
<td>A date or date range and possibly role associated with the physical object. For example: &quot;Illustrated c. 1200&quot;</td>
</tr>
<tr>
<td>sc:locationLabel</td>
<td>Property</td>
<td>A location and possibly role associated with the physical object. For example: &quot;Paris, France (created)&quot;</td>
</tr>
<tr>
<td>sc:attributionLabel</td>
<td>Property</td>
<td>An attribution that must be displayed along with the resource. For example: &quot;Held at A Library (NY)&quot;</td>
</tr>
<tr>
<td>sc:rightsLabel</td>
<td>Property</td>
<td>A rights or license statement, describing how the facsimile may be reused.</td>
</tr>
<tr>
<td>dc:description</td>
<td>Property</td>
<td>A longer form description of the object. It may be associated with any of the resources in the model.</td>
</tr>
</tbody>
</table>

Services and Bibliographic Information Model:
Figure 5.2. Manifest Model

```xml
<Manifest1> a sc:Manifest, ore:Aggregation ;
  ore:aggregates <Seq1>, <Range1> ;
  rdfs:label "Title" ;
  sc:agentLabel "Name" ;
  sc:dateLabel "Date" ;
  sc:hasRelatedService <Service1> ;
  sc:hasRelatedDescription <BibDesc1> .
```

Recommendations

- http://library.stanford.edu/iiif/image-api/
- http://www.shared-canvas.org/datamodel/search/

A. References

[Creative Commons] Creative Commons, Creative Commons, May 18 2008

[Content in RDF] Representing Content in RDF 1.0, Koch, J. Velasco, C. 29 October 2009


[DC Terms] DCMI Metadata Terms, DCMI Usage Board, 18 December 2006.


B. Acknowledgements

The editors would like to acknowledge the financial support of the Mellon Foundation.
C. Change Log

<table>
<thead>
<tr>
<th>Date</th>
<th>Editor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-05-01</td>
<td>rsanderson</td>
<td>Internal Draft 1</td>
</tr>
</tbody>
</table>

This work by the Open Annotation Collaboration is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
Abstract

Annotations are typically used to convey information about a resource or associations between resources. Simple examples include a comment or tag on a single web page or image, or a blog post about a news article.

The Web Annotation Data Model specification describes a structured model and format to enable annotations to be shared and reused across different hardware and software platforms. Common use cases can be modeled in a manner that is simple and convenient, while at the same time enabling more complex requirements, including linking arbitrary content to a particular data point or to segments of timed multimedia resources.

The specification provides a specific JSON format for ease of creation and consumption of annotations based on the conceptual model that accommodates these use cases, and the vocabulary of terms that represents it.
Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This specification was derived from the Open Annotation Community Group's outcomes, and details of the differences between the two are maintained in the Acknowledgment appendix.

This document was published by the Web Annotation Working Group as a Recommendation. If you wish to make comments regarding this document, please send them to public-annotation@w3.org (subscribe, archives). All comments are welcome.

Please see the Working Group's implementation report.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 September 2015 W3C Process Document.

Table of Contents

1. Introduction
   1.1 Aims of the Model
   1.2 Serialization of the Model
   1.3 Conformance
      1.3.1 Conformance Requirements Related to Selectors
   1.4 Terminology

2. Web Annotation Principles

3. Web Annotation Framework
   3.1 Annotations
   3.2 Bodies and Targets
      3.2.1 External Web Resources
      3.2.2 Classes
      3.2.3 Segments of External Resources
      3.2.4 Embedded Textual Body
      3.2.5 String Body
      3.2.6 Cardinality of Bodies and Targets
      3.2.7 Choice Between Bodies
   3.3 Other Properties
3.3.1 Lifecycle Information
3.3.2 Agents
3.3.3 Intended Audience
3.3.4 Accessibility of Content
3.3.5 Motivation and Purpose
3.3.6 Rights Information
3.3.7 Other Identities

4. Specific Resources
4.1 Purpose for External Web Resources
4.2 Selectors
  4.2.1 Fragment Selector
  4.2.2 CSS Selector
  4.2.3 XPath Selector
  4.2.4 Text Quote Selector
  4.2.5 Text Position Selector
  4.2.6 Data Position Selector
  4.2.7 SVG Selector
  4.2.8 Range Selector
  4.2.9 Refinement of Selection
4.3 States
  4.3.1 Time State
  4.3.2 Request Header State
  4.3.3 Refinement of State
4.4 Styles
4.5 Rendering Software
4.6 Scope of a Resource

5. Collections
5.1 Annotation Collection
5.2 Annotation Page

A. Correspondence Among Media Types and Selectors
A.1 Additional Media Types/Selector Combination

B. Complete Example

C. Index of JSON Keys

D. Sets of Bodies and Targets

E. Acknowledgments

F. Candidate Recommendation Exit Criteria

G. Changes from Previous Versions
G.1 Changes from the Proposed Recommendation of 2017-01-17
G.2 Changes from the Candidate Recommendation of 2016-11-22
1. Introduction

This section is non-normative.

Annotating, the act of creating associations between distinct pieces of information, is a pervasive activity online in many guises. Web citizens make comments about online resources using either tools built in to the hosting website, external web services, or the functionality of an annotation client. Comments about shared photos or videos, reviews of products, or even social network mentions of web resources could all be considered as annotations. In addition, there are a plethora of "sticky note" systems and stand-alone multimedia annotation systems. This specification describes a common approach to expressing these annotations, and more.

The Web Annotation Data Model provides an extensible, interoperable framework for expressing annotations such that they can easily be shared between platforms, with sufficient richness of expression to satisfy complex requirements while remaining simple enough to also allow for the most common use cases, such as attaching a piece of text to a single web resource.

An annotation is considered to be a set of connected resources, typically including a body and target, and conveys that the body is related to the target. The exact nature of this relationship changes according to the intention of the annotation, but the body is most frequently somehow "about" the target. This perspective results in a basic model with three parts, depicted below. The full model supports additional functionality, enabling content to be embedded within the annotation, selecting arbitrary segments of resources, choosing the appropriate representation of a resource and providing styling hints to help clients render the annotation appropriately. Annotations created by or intended for machines are also possible, ensuring that the Data Web is not ignored in favor of only considering the human-oriented Document Web.

The Web Annotation Data Model does not prescribe a transport protocol for creating, managing and retrieving annotations. Instead it describes a resource oriented structure and serialization of that structure that could be carried over many different protocols. The related [annotation-protocol] specification describes a recommended transport layer, which may be adopted separately.
1.1 Aims of the Model

The primary aim of the Web Annotation Data Model is to provide a standard description model and format to enable annotations to be shared between systems. This interoperability may be either for sharing with others, or the migration of private annotations between devices or platforms. The shared annotations must be able to be integrated into existing collections and reused without loss of significant information. The model should cover as many annotation use cases as possible, while keeping the simple annotations easy and expanding from that baseline to make complex uses possible.

The Web Annotation Data Model is a single, consistent model that can be used by all interested parties. All efforts have been made to keep the implementation costs for both producers and consumers to a minimum. A single method of fulfilling a use case is strongly preferred over multiple methods, unless there are existing standards that need to be accommodated or there is a significant cost associated with using only a single method. While the Data Model is built using Linked Data fundamentals, the design is intended to allow rich and performant non-graph-based implementations. As such, inferencing and other graph-based queries are explicitly not a priority for optimization in the design of the model.

1.2 Serialization of the Model

The examples throughout the document are serialized as [JSON-LD] using the Context given in Appendix A of the Annotation Vocabulary [annotation-vocab], which is the preferred serialization format. The media type of this format is defined in Section 3 of the Annotation Protocol [annotation-protocol] as application/ld+json;profile="http://www.w3.org/ns/anno.jsonld".

When the only information that is recorded in the annotation is the IRI of a resource, then that IRI is used as the value of the relationship, as in Example 1. When there is more information about the resource, the IRI is the value of the id property of the object which is the value of the relationship, as in Example 2.

1.3 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words may, must, must not, not recommended, recommended, should, and should not are to be interpreted as described in [RFC2119].

1.3.1 Conformance Requirements Related to Selectors

Not all Selectors are relevant for all media types; some combinations are meaningless or not formally defined. An implementation may therefore ignore certain types of Selectors in case the corresponding media types are not handled by that particular implementation.

The table in A. Correspondence Among Media Types and Selectors shows the correspondence among the main media types addressed in this specification and Selector types. The meaning of the table elements, and their effect on implementation conformance, is as follows.

- A “✔” sign in the table means that the Selector type is relevant for that particular media types. A conforming implementation must implement that particular combination if it handles the corresponding media type.
A “✘” sign in the table means that the Selector type is not relevant for that particular media types. Conforming implementations should ignore that particular combination.

A “?” means that it is not possible to specify, in general, whether that particular combination is meaningful (e.g., fragment identifiers are defined for specific media types, i.e., specific text media types other than plain text may have fragments defined but they are not listed in the table). In other cases the usefulness of such combination is not clear (e.g., Data Position selectors for binary images). Conforming implementations may implement that particular combination.

1.4 Terminology

IRI
An IRI, or Internationalized Resource Identifier, is an extension to the URI specification to allow characters from Unicode, whereas URIs must be made up of a subset of ASCII characters. There is a mapping algorithm for translating between IRIs and the equivalent encoded URI form. IRIs are defined by [rfc3987].

Resource
An item of interest that may be identified by an IRI.

Web Resource
A Resource that must be identified by an IRI, as described in the Web Architecture [webarch]. Web Resources may be dereferencable via their IRI.

External Web Resource
A Web Resource which is not part of the representation of the Annotation, such as a web page, image, or video. External Web Resources are dereferencable from their IRI.

Property
A feature of a Resource, that often has a particular data type. In the model sections, the term "Property" is used to refer to only those features which are not Relationships and instead have a literal value such as a string, integer or date. The valid values for a Property are thus any data type other than object, or an array containing members of that data type if more than one is allowed.

Relationship
In the model sections, the term "Relationship" is used to distinguish those features that refer to other Resources, either by reference to the Resource's IRI or by including a description of the Resource in the Annotation's representation. The valid values for a Relationship are: a quoted string containing an IRI, an object that has the "id" property, or an array containing either of these if more than one is allowed.

Class
Resources may be divided, conceptually, into groups called "classes"; members of a class are known as Instances of the class. Resources are associated with a particular class through typing. Classes are identified by IRIs, i.e., they are also Web Resources themselves.

Type
A special Relationship that associates an Instance of a class to the Class it belongs to.

Instance
An element of a group of Resources represented by a particular Class.

2. Web Annotation Principles

The Web Annotation Data Model is defined using the following basic principles:

- An Annotation is a rooted, directed graph that represents a relationship between resources.
There are two primary types of resource that participate in this relationship, Bodies and Targets.

- Annotations have 0 or more Bodies.
- Annotations have 1 or more Targets.
- The content of the Body resources is related to, and typically "about", the content of the Target resources.
- Annotations, Bodies and Targets may have their own properties and relationships, typically including creation and descriptive information.
- The intent behind the creation of an Annotation or the inclusion of a particular Body or Target is an important property and represented by a Motivation resource.

The following principles describe additional distinctions regarding the exact nature of Target and Body:

- The Target or Body resource may be more specific than the entity identified by the resource's IRI alone.
- In particular,
  - The Target or Body resource may be a specific segment of the resource.
  - The Target or Body resource may be styled in a specific way.
  - The Target or Body resource may be a specific state of the resource.
  - The Target or Body resource may be included in the Annotation to play a specific role.
  - The Target or Body resource may be any combination of the above.
- The resource with these constraints is a separate resource from the Annotation, Body or Target, and is called a SpecificResource.
- The SpecificResource refers to the source resource and the constraints that make it more specific.
- The identity of the SpecificResource is separate from the descriptions of the constraints.
- The Body resource may be a choice between multiple resources.

The properties of external resources, such as Bodies and Targets, included in the Annotation document are intended as hints to the client, and are not to be considered authoritative information. This includes properties such as the created time, the creating agent, the modification time, any rights assertions, format, language or text direction of the external resource.

3. Web Annotation Framework

3.1 Annotations

An Annotation is a Web Resource. Typically, an Annotation has a single Body, which is a comment or other descriptive resource, and a single Target that the Body is somehow "about". The Annotation likely also has additional descriptive properties.

**Example Use Case:** Alice has written a post that makes a comment about a particular web page. Her client creates an Annotation with the post as the body resource, and the web page as the target resource.

Model
<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td>Property</td>
<td>The context that determines the meaning of the JSON as an Annotation. The Annotation must have 1 or more @context values and <a href="http://www.w3.org/ns/anno.jsonld">http://www.w3.org/ns/anno.jsonld</a> must be one of them. If there is only one value, then it must be provided as a string.</td>
</tr>
<tr>
<td>id</td>
<td>Property</td>
<td>The identity of the Annotation. An Annotation must have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the Annotation. An Annotation must have 1 or more types, and the Annotation class must be one of them.</td>
</tr>
<tr>
<td>Annotation</td>
<td>Class</td>
<td>The class for Web Annotations. The Annotation class must be associated with an Annotation using type.</td>
</tr>
<tr>
<td>body</td>
<td>Relationship</td>
<td>The relationship between an Annotation and its Body. There should be 1 or more body relationships associated with an Annotation but there may be 0.</td>
</tr>
<tr>
<td>target</td>
<td>Relationship</td>
<td>The relationship between an Annotation and its Target. There must be 1 or more target relationships associated with an Annotation.</td>
</tr>
</tbody>
</table>

**Example**

**EXAMPLE 1: Basic Annotation Model**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno1",
    "type": "Annotation",
    "body": "http://example.org/post1",
    "target": "http://example.com/page1"
}
```

### 3.2 Bodies and Targets

The Web is distributed, with different systems working together to provide access to content. Annotations can be used to link those resources together, being referenced as the Body and Target. The Target resource is always an External Web Resource, but the Body may also be embedded within the Annotation. External Web Resources may be separately dereferenced to retrieve a representation of their state, whereas the embedded Body does not need to be dereferenced as the representation is included within the Annotation's representation.

#### 3.2.1 External Web Resources
Web Resources are identified with a IRI and have various properties, often including a format or language for the resource's content. This information may be recorded as part of the Annotation, even if the representation of the resource must be retrieved from the Web.

**Example Use Case:** Beatrice records a long analysis of a patent, and publishes the audio on her website as an mp3. She then creates an Annotation with the mp3 as the body, and the PDF of the patent as the target.

## Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the Body or Target resource. Bodies or Targets which are External Web Resources must have exactly 1 id with the value of the resource’s IRI.</td>
</tr>
<tr>
<td>format</td>
<td>Property</td>
<td>The format of the Web Resource’s content. The Body or Target should have exactly 1 format associated with it, but may have 0 or more. The value of the property should be the media-type of the format, following the [rfc6838] specification.</td>
</tr>
<tr>
<td>language</td>
<td>Property</td>
<td>The language of the Web Resource’s content. The Body or Target should have exactly 1 language associated with it, but may have 0 or more, for example if the language cannot be identified or the resource contains a mix of languages. The value of the property should be a language code following the [bcp47] specification.</td>
</tr>
<tr>
<td>processingLanguage</td>
<td>Property</td>
<td>The language to use for text processing algorithms such as line breaking, hyphenation, which font to use, and similar. Each Body and Target may have exactly 1 processingLanguage. The value of the property should be a language code following the [bcp47] specification. If this property is not present and the language property is present with a single value, then the client should use that language for processing requirements.</td>
</tr>
<tr>
<td>textDirection</td>
<td>Relationship</td>
<td>The overall base direction of the text in the resource. The Body or Target may have exactly 1 textDirection associated with it. The value of the property must be one of the directions defined below (ltr, rtl, or auto).</td>
</tr>
<tr>
<td>ltr</td>
<td>Instance</td>
<td>The direction that indicates the value of the resource is explicitly directionally isolated left-to-right text.</td>
</tr>
<tr>
<td>rtl</td>
<td>Instance</td>
<td>The direction that indicates the value of the resource is explicitly directionally isolated right-to-left text.</td>
</tr>
<tr>
<td>auto</td>
<td>Instance</td>
<td>The direction that indicates the value of the resource is explicitly directionally isolated text, and the direction is to be programmatically determined using the value.</td>
</tr>
</tbody>
</table>
3.2.2 Classes

It is useful for clients to know the general type of a Web Resource in advance. If the client cannot render videos, then knowing that the Body is a video will allow it to avoid needlessly downloading a potentially large content stream. For resources that do not have obvious media types, such as many data formats, it is also useful for a client to know that a resource with the format text/csv should not simply be rendered as plain text, despite the first part of the media type, whereas application/pdf may be able to be rendered by the user agent despite the main type being ‘application’.

**Example Use Case:** Corina shoots a video of her comment about a website on her phone and uploads it. She associates the video with the website via an Annotation, and her client adds types as a hint to consuming systems.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>

**NOTE**

The [iana-media-types](https://www.iana.org/assignments/media-types) document provides the official registry of media types that can be used with the format property. The [w3c-language-tags](https://www.w3.org/International/articles/language-tags/) article provides a good overview of the values that implementers can expect to encounter in the language property. The notion of text direction and the definitions of auto, ltr and rtl values are taken explicitly from the HTML5 [html5](https://html5doctor.com/) dir attribute. Please also note that if information provided by the external resource contradicts the information provided by the annotation about it, then the external resource is authoritative and the information from the annotation should be disregarded.

**Example 2: External Web Resources**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno2",
    "type": "Annotation",
    "body": {
        "id": "http://example.org/analysis1.mp3",
        "format": "audio/mpeg",
        "language": "fr"
    },
    "target": {
        "id": "http://example.gov/patent1.pdf",
        "format": "application/pdf",
        "language": ["en", "ar"],
        "textDirection": "ltr",
        "processingLanguage": "en"
    }
}
```
### 3.2.3 Segments of External Resources

Many Annotations involve part of an External Web Resource, rather than its entirety. In the Web [webarch], segments of resources are identified using IRIs with a fragment component that at the same time both describes how to extract the segment of interest from the resource, and identifies the extracted content. For simple Annotations, it is valuable to be able to use these IRIs with a fragment component as the identifier for either Body or Target.

**Example Use Case:** Dawn wants to describe a particular region of an image. She highlights that area in her client and types in the description. Her client then constructs an IRI with an appropriate fragment component as the target.

### Example

**EXAMPLE 3: Typing of Body and Target**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno3",
    "type": "Annotation",
    "body": {
        "id": "http://example.org/video1",
        "type": "Video"
    },
    "target": {
        "id": "http://example.org/website1",
        "type": "Text"
    }
}
```

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the Body or Target resource. Bodies or Targets which are External Web Resources must have exactly 1 id with the value of</td>
</tr>
</tbody>
</table>
the resource's IRI, and that IRI may have a fragment component.

It is important to be aware of the consequences of using an IRI with a fragment component, and the restrictions that using them places on implementations.

- Fragments are defined with respect to individual media types. For example, HTML has a specific set of semantics regarding the meaning of the fragment part of the IRI.
- Not every media type has a fragment specification. For example, Office documents might have a media-type and be published on the web, but not have semantics associated with the fragment part of the IRI.
- Even if a media type does have a fragment definition, it is often not possible to describe the segment of interest sufficiently precisely. For example, fragments for HTML cannot be used to describe an arbitrary range of text.
- It is not possible to determine with certainty what is being identified without knowing the media type, as the same fragment string might be possible in different specifications. For example, the same fragment string could identify either a rectangular area in an image, or a strangely named section of an HTML document.
- IRIs with a fragment component are not compatible with other methods of describing the segment more specifically. For example, it is not possible to describe how to retrieve the correct representation, add style information, or associate a role with the resource, using such IRIs. The method to accomplish these requirements is described in the Fragment Selector portion of the Specific Resources section.
- As IRIs are considered to be opaque strings, annotation systems may not discover annotations with fragment components when searching by means of the IRI without the fragment. For example, an Annotation with the Target http://example.com/image.jpg#xywh=1,1,1,1 would not be discovered in a simple search for http://example.com/image.jpg, even though it is part of it.

For more information regarding the use of IRIs with fragment components, please see the Best Practices for Fragment Identifiers and Media Type Definitions [fragid-best-practices].

Example

```
EXAMPLE 4: IRIs with Fragment Components

{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno4",
    "type": "Annotation",
    "body": "http://example.org/description1",
    "target": {
        "id": "http://example.com/image1#xywh=100,100,300,300",
        "type": "Image",
        "format": "image/jpeg"
    }
}
```
3.2.4 Embedded Textual Body

In many situations, the Body of the Annotation will be in a text format, and created at the same time as the Annotation without a separate IRI. In these cases, the Body’s text can be included as part of the Annotation to avoid having to interact with multiple systems. The Body may also have the features of External Web Resources, including especially the language of the text and the format that it is conveyed in.

**Example Use Case:** Emily writes a comment about how much she likes an image on a photo sharing website. Her client creates an Annotation with the comment embedded within it, and adds that it is in French and formatted using HTML.

**Model**

The fundamental features of a Textual Body are:

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the Textual Body. The Body may have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the Textual Body resource. The Body should have the <code>TextualBody</code> class, and may have other classes.</td>
</tr>
<tr>
<td>TextualBody</td>
<td>Class</td>
<td>A class assigned to the Body for embedding textual resources within the Annotation. The Body should have the <code>TextualBody</code> class.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The character sequence of the content of the Textual Body. There must be exactly 1 value property associated with the TextualBody.</td>
</tr>
</tbody>
</table>

Systems should assume that Textual Bodies have the `Text` class, described in **Classes** above, even if it is not explicitly included in the `type` property.

The properties of External Web Resources, such as `language` and `format` also apply to embedded Textual Body resources.

**Example**

```
EXAMPLE 5: Textual Body

{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno5",
    "type": "Annotation",
    "body": {
        "type": "TextualBody",
        "value": "<p>j'adore !</p>",
        "format": "text/html",
        "language": "fr"
    },
    "target": "http://example.org/photo1"
}
```
3.2.5 String Body

The simplest type of Body is a plain text string, without additional information or properties. This type of Body is useful for the simplest of Annotations only, and is not recommended for uses where the Body may need to be referred to from outside of the Annotation.

Example Use Case Franceska wants to create a quick Annotation from a simple, command line client. She creates the JSON serialization in a text file and sends it to her Annotation server to maintain.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bodyValue</td>
<td>Property</td>
<td>The string value of the Body of the Annotation. There may be exactly 1 bodyValue for an Annotation, and the value must conform to the requirements below. If the bodyValue property is present, then the body relationship must not also be present.</td>
</tr>
</tbody>
</table>

There are several restrictions on when this form may be used and how it is to be interpreted. The string Body:

- must be a single xsd:string and the data type must not be expressed in the serialization.
- must not have a language associated with it.
- must be interpreted as if it were the value of the value property of a Textual Body.
- must be interpreted as if the Textual Body resource had a format property with the value text/plain.
- must not have the value of other properties of the Textual Body inferred from similar properties on the Annotation resource.

If any of the interpretations above are not correct, then the TextualBody construction must be used instead.

NOTE

Systems may rewrite Annotations to instead use the TextualBody construction, rather than maintaining the bodyValue form. The TextualBody construction is preferred, as language and format information are important for clients processing the Annotation.

Example
3.2.6 Cardinality of Bodies and Targets

Some Annotations may not have a Body at all, such as a simple highlight or bookmark without any accompanying text. It is also possible for an Annotation to have multiple Bodies and/or Targets. In this case, each Body is considered to be equally related to each Target individually, rather than to the set of all of the Targets.

**Example Use Case:** Gretchen highlights a particular region of her ebook in green and, knowing what such a highlight means, she does not give a comment. Her client associates a stylesheet with the Annotation, and does not create a body at all.

**Example Use Case:** Hannah associates a tag and a description with two images using a single annotation.

**Model**

The body relationship is omitted when there is no Body for the Annotation.

The body and/or target relationships of the Annotation may be arrays rather than a single object. The values may be either strings containing the IRI of the resource or objects.

**Example**
3.2.7 Choice Between Bodies

A Choice has an ordered list of resources from which an application should select only one to process or display. The order is given from the most preferable to least preferable, according to the Annotation's creator or publisher.

**Example Use Case:** Irina writes up her discussion of a particular website in both French and English. As the two posts are equivalent, there is no need to display both, and instead she wants French speakers to see the French comment, and everyone else to see the English version. Her client creates as Choice with the English comment listed first.

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the Choice.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Choice <strong>may</strong> have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the resource.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Choice <strong>must</strong> have exactly 1 <strong>type</strong>, and it <strong>must</strong> be the CHOICE class.</td>
</tr>
<tr>
<td>Choice</td>
<td>Class</td>
<td>A construction that conveys to a consuming application that it <strong>should</strong> select one of the</td>
</tr>
</tbody>
</table>
listed resources to display to the user, and not render all of them.

| items | Relationship | A list of resources to choose from, with the default option listed first. |

**NOTE**
Clients may use any algorithm to determine which resource to choose, and should make use of the information present to do so automatically, but may present a list and require the user to make the decision.

**Example**

**EXAMPLE 10: Choice**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno10",
    "type": "Annotation",
    "body": {
        "type": "Choice",
        "items": [
            {
                "id": "http://example.org/note1",
                "language": "en"
            },
            {
                "id": "http://example.org/note2",
                "language": "fr"
            }
        ]
    },
    "target": "http://example.org/website1"
}
```

### 3.3 Other Properties

It is often important to have information about the context in which the Annotation and any External Web Resources were created, modified and used. In particular,

- When was the resource created, modified or generated
- Who created, modified or generated the serialized form of the Annotation or other resource, and who is it intended for
- Why was the resource included in the annotation, or the annotation created
- What other identities the resource has
- How can the resource be used, according to its rights and licensing
### 3.3.1 Lifecycle Information

The person, organization or machine responsible for the Annotation or referenced resource deserves credit for their contribution, and the time at which those resources are created is useful for display ordering and filtering out old, potentially irrelevant content. The creator of the Annotation is also useful for determining the trustworthiness of the Annotation. The software used to create and serialize the Annotation, along with when that activity occurred, is useful for both advertising and debugging issues.

**Example Use Case:** Jane writes a review of a restaurant online, and wishes to be associated with that review so that her friends know that it was her review and can trust it. Her client adds her account’s identity, and its own identity, to the Annotation and the appropriate timestamps for when the resources were created.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>creator</td>
<td>Relationship</td>
<td>The agent responsible for creating the resource. This may be either a human, an organization or a software agent. There should be exactly 1 creator relationship for Annotation and Body, but may be 0 or more than 1, as the resource’s creator may wish to remain anonymous, or multiple agents may have worked together on it. The relationships may be associated with other resources.</td>
</tr>
<tr>
<td>created</td>
<td>Property</td>
<td>The time at which the resource was created. There should be exactly 1 created property for Annotation and Body, and must not be more than 1. The property may be associated with other resources. The datetime must be a xsd:dateTime with the UTC timezone expressed as &quot;Z&quot;.</td>
</tr>
<tr>
<td>generator</td>
<td>Relationship</td>
<td>The agent responsible for generating the serialization of the Annotation. There may be 0 or more generator relationships per Annotation</td>
</tr>
<tr>
<td>generated</td>
<td>Property</td>
<td>The time at which the Annotation serialization was generated. There may be exactly 1 generated property per Annotation, and must not be more than 1. The datetime must be a xsd:dateTime with the UTC timezone expressed as &quot;Z&quot;.</td>
</tr>
<tr>
<td>modified</td>
<td>Property</td>
<td>The time at which the resource was modified, after creation. There may be exactly 1 modified property for Annotation and Body, and must not be more than 1. The property may be associated with other resources. The datetime must be a xsd:dateTime with the UTC timezone expressed as &quot;Z&quot;.</td>
</tr>
</tbody>
</table>

**NOTE**

Beyond the features described in this section, other properties may be added features of the Annotation or any resource in the model. Please see the Extension section of [annotation-vocab] for more information about how to do this.
3.3.2 Agents

More information about the agents involved in the creation of an Annotation is normally required beyond an IRI that identifies them. This includes whether they are an individual, a group or a piece of software and properties such as real name, account nickname, and email address.

Example Use Case: Kelly wants to submit an Annotation to a system that does not manage her identity, and would like a pseudonym to be displayed. Her client adds this information to the Annotation to send to the service.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the agent. An Agent <strong>should</strong> have exactly 1 IRI that identifies it, and <strong>must not</strong> have more than 1.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the Agent. An Agent <strong>should</strong> have 1 or more classes, from those listed below.</td>
</tr>
<tr>
<td>Person</td>
<td>Class</td>
<td>The class for a human agent.</td>
</tr>
<tr>
<td>Organization</td>
<td>Class</td>
<td>The class for an organization, as opposed to an individual.</td>
</tr>
<tr>
<td>Software</td>
<td>Class</td>
<td>The class for a software agent, such as a user's client or a machine learning system that creates Annotations.</td>
</tr>
<tr>
<td>name</td>
<td>Property</td>
<td>The name of the agent. Each agent <strong>should</strong> have exactly 1 name property, and <strong>may</strong> have 0 or more.</td>
</tr>
<tr>
<td>nickname</td>
<td>Property</td>
<td>The nickname of the agent. Each agent should have exactly 1 nickname property, and may have 0.</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>email</td>
<td>Relationship</td>
<td>The email address associated with the agent, using the mailto: IRI scheme [rfc6086]. Each agent may have 1 or more email addresses.</td>
</tr>
<tr>
<td>email_sha1</td>
<td>Property</td>
<td>The text representation of the result of applying the sha1 algorithm to the email IRI of the agent, including the 'mailto:' prefix and no whitespace. This allows the mail address to be used as an identifier without publishing the address publicly. Each agent may have 1 or more values in the email_sha1 property.</td>
</tr>
<tr>
<td>homepage</td>
<td>Relationship</td>
<td>The home page for the agent. Each agent may have 1 or more home pages.</td>
</tr>
</tbody>
</table>

Example

**EXAMPLE 12: Agents**

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno12",
  "type": "Annotation",
  "creator": {
    "id": "http://example.org/user1",
    "type": "Person",
    "name": "My Pseudonym",
    "nickname": "pseudo",
    "email_sha1": "58bad08927902ff9307b621c54716dccc5083e339"
  },
  "generator": {
    "id": "http://example.org/client1",
    "type": "Software",
    "name": "Code v2.1",
    "homepage": "http://example.org/client1/homepage1"
  },
  "body": "http://example.net/review1",
  "target": "http://example.com/restaurant1"
}
```

### 3.3.3 Intended Audience

Beyond the agents that are associated with the creation and management of the Annotation and other resources, it is also useful to know the audience or class of consuming agent that the resource is intended for. This allows for the roles (such as teacher versus student) or properties of the class (such as a suggested age range) of the intended audience to be recorded.

**Example Use Case:** Lynda writes some notes about using a particular textbook to teach a class. She adds that the intended audience of the Annotation is teachers (who are using the textbook), to distinguish from other Annotations that might have
an audience of the students (who are also using the textbook, but to learn from).

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the Audience. There may be exactly 1 IRI given that identifies the Audience.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the Audience, from the schema.org class structure. The Audience should have 1 or more types and they should come from the schema.org class structure.</td>
</tr>
<tr>
<td>audience</td>
<td>Relationship</td>
<td>The relationship between an Annotation and its intended Audience. There may be 0 or more Audiences for each Annotation.</td>
</tr>
</tbody>
</table>

Further properties that describe the audience are used from schema.org's Audience classes. The properties and class names must be prefixed in the JSON with schema: to ensure that they are uniquely distinguished from any other properties or classes.

The use of audience does not imply nor enable any access restriction to prevent the annotation from being seen. Systems should use the information for filtering the display of Annotations based on their knowledge of the user, and not assume that the Annotation or other resources will require authentication and authorization.

#### Example

**EXAMPLE 13: Audience**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno13",
    "type": "Annotation",
    "audience": {
        "id": "http://example.edu/roles/teacher",
        "type": "schema:EducationalAudience",
        "schema:educationalRole": "teacher"
    },
    "body": "http://example.net/classnotes1",
    "target": "http://example.com/textbook1"
}
```

#### 3.3.4 Accessibility of Content

Access to information is recognized as a basic human right by the United Nations. The Web is able to remove barriers to communication and interaction regardless of various physical impediments. This supports social inclusion, but also increases the potential audience of the information. For resources that are used as the Body or Target of an Annotation, it is valuable to record the features of that resource that provide easier access for users with various and diverse ranges of ability.
**Example Use Case:** Megan has very limited ability to hear sound, and prefers to read captions when interacting with videos. She uses her annotation client to make a comment on such a video, and to help others in the same situation, the client includes that the video has this accessibility feature.

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessibility</td>
<td>Property</td>
<td>One or more strings from an enumerated list of values that each describes an accessibility feature that the resource has. There may be 0 or more accessibility features listed for each Body or Target resource.</td>
</tr>
</tbody>
</table>

**NOTE**

The current list of values is referenced from the [schema.org description](http://schema.org/accessibilityFeature) of the accessibilityFeature property.

**Example**

**EXAMPLE 14: Accessibility**

```json
{
   "@context": "http://www.w3.org/ns/anno.jsonld",
   "id": "http://example.org/anno14",
   "type": "Annotation",
   "motivation": "commenting",
   "body": "http://example.net/comment1",
   "target": {
      "id": "http://example.com/video1",
      "type": "Video",
      "accessibility": "captions"
   }
}
```

**3.3.5 Motivation and Purpose**

In many cases it is important to understand the reasons why the Annotation was created, or why the Textual Body was included in the Annotation, not just the times and agents involved. These reasons are provided by declaring the motivation for the Annotation's creation or the purpose for the inclusion of the Textual Body in the Annotation; the "why" rather than the "who" and "when" described in the previous sections.

**Example Use Case:** Noelle annotates a resource intending to bookmark it for future reference, and provides a description and a tag to make it easier to find again. Her client adds the right motivations to the Annotation and the Textual Body resources to capture this.
<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>motivation</td>
<td>Relationship</td>
<td>The relationship between an Annotation and a Motivation. There should be exactly 1 motivation for each Annotation, and may be 0 or more than 1.</td>
</tr>
<tr>
<td>purpose</td>
<td>Relationship</td>
<td>The reason for the inclusion of the Textual Body within the Annotation. There may be 0 or more purposes associated with a TextualBody.</td>
</tr>
<tr>
<td>Motivation</td>
<td>Class</td>
<td>The Motivation for an Annotation is a reason for its creation, and might include things like Replying to another annotation, Commenting on a resource, or Linking to a related resource.</td>
</tr>
</tbody>
</table>

### Motivations

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assessing</td>
<td>Instance</td>
<td>The motivation for when the user intends to assess the target resource in some way, rather than simply make a comment about it. For example to write a review or assessment of a book, assess the quality of a dataset, or provide an assessment of a student's work.</td>
</tr>
<tr>
<td>bookmarking</td>
<td>Instance</td>
<td>The motivation for when the user intends to create a bookmark to the Target or part thereof. For example an Annotation that bookmarks the point in a text where the reader finished reading.</td>
</tr>
<tr>
<td>classifying</td>
<td>Instance</td>
<td>The motivation for when the user intends to classify the Target as something. For example to classify an image as a portrait.</td>
</tr>
<tr>
<td>commenting</td>
<td>Instance</td>
<td>The motivation for when the user intends to comment about the Target. For example to provide a commentary about a particular PDF document.</td>
</tr>
<tr>
<td>describing</td>
<td>Instance</td>
<td>The motivation for when the user intends to describe the Target, as opposed to (for example) a comment about it. For example describing the above PDF's contents, rather than commenting on their accuracy.</td>
</tr>
<tr>
<td>editing</td>
<td>Instance</td>
<td>The motivation for when the user intends to request a change or edit to the Target resource. For example an Annotation that requests a typo to be corrected.</td>
</tr>
<tr>
<td>highlighting</td>
<td>Instance</td>
<td>The motivation for when the user intends to highlight the Target resource or segment of it. For example to draw attention to the selected text that the annotator disagrees with.</td>
</tr>
<tr>
<td>identifying</td>
<td>Instance</td>
<td>The motivation for when the user intends to assign an identity to the Target. For example to associate the IRI that identifies a city with a mention of the city in a web page.</td>
</tr>
<tr>
<td>linking</td>
<td>Instance</td>
<td>The motivation for when the user intends to link to a resource related to the Target.</td>
</tr>
<tr>
<td>moderating</td>
<td>Instance</td>
<td>The motivation for when the user intends to assign some value or quality to the Target. For example annotating an Annotation to moderate it up in a trust network or threaded discussion.</td>
</tr>
<tr>
<td>questioning</td>
<td>Instance</td>
<td>The motivation for when the user intends to ask a question about the Target. For example to ask for assistance with a particular section of text, or question its...</td>
</tr>
</tbody>
</table>
### 3.3.6 Rights Information

It is common practice to associate a license or rights statement with a resource, in order to describe the conditions under which it may be used. This allows the user to make appropriate use of the resource, as well as allowing some automated systems to confirm that the usage is permitted. As the Annotation, Bodies, and Targets might be created with different licences or rights, each can be described separately. The rights of resources other than the Annotation itself are considered informative hints to a consuming client application.

**Example Use Case:** Ophelia writes a review of a product and wishes to be known as the author of the review, however does not mind how the Annotation that relates the review and the product together is used. She asserts these two separate rights

<table>
<thead>
<tr>
<th>Veracity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replying Instance</td>
<td>The motivation for when the user intends to reply to a previous statement, either an Annotation or another resource. For example providing the assistance requested in the above.</td>
</tr>
<tr>
<td>tagging Instance</td>
<td>The motivation for when the user intends to associate a tag with the Target.</td>
</tr>
</tbody>
</table>

**NOTE**

For more information about how Motivations can be inter-related and new Motivations created, please see the Annotation Vocabulary document [annotation-vocab]. Section 4.1 describes how to associate a Motivation with and external web resource.

### Example

**EXAMPLE 15: Motivation and Purpose**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno15",
    "type": "Annotation",
    "motivation": "bookmarking",
    "body": [
        {
            "type": "TextualBody",
            "value": "readme",
            "purpose": "tagging"
        },
        {
            "type": "TextualBody",
            "value": "A good description of the topic that bears further investigation",
            "purpose": "describing"
        }
    ],
    "target": "http://example.com/page1"
}
```
statements with the Annotation and Body individually. She does not know the rights asserted on the target resource, so does not specify any.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rights</td>
<td>Relationship</td>
<td>The relationship between an Annotation, Body or Target and a Web Resource that contains the rights statement or license under which the resource may be used. There may be at exactly 0 or more rights statements or licenses linked from each resource, and the value must be an IRI.</td>
</tr>
</tbody>
</table>

Example

EXAMPLE 16: Rights

```
{
   "@context": "http://www.w3.org/ns/anno.jsonld",
   "id": "http://example.org/anno16",
   "type": "Annotation",
   "rights": "https://creativecommons.org/publicdomain/zero/1.0/",
   "body": {
      "id": "http://example.net/review1",
      "rights": "http://creativecommons.org/licenses/by-nc/4.0/"
   },
   "target": "http://example.com/product1"
}
```

3.3.7 Other Identities

In a massively distributed system such as the Web, information is often copied. In order to track the provenance of the Annotation and other related resources, it is possible to record additional IRIs that also identify the resource. These may be dereferencable "permalinks", identities assigned by a client offline without any knowledge of the web, or simply the location where the current harvesting system discovered the resource.

Example Use Case: Petra creates an Annotation and sends it to multiple systems to maintain, one personal and one public. She wants to ensure that the copies can be aligned, and so she sets a UUID as the canonical IRI, allowing the service to assign an HTTP IRI for it. A subsequent system then harvests the public copy, maintaining the canonical UUID as discovered, then moves the original HTTP IRI to via, replacing it with an IRI under its control.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canonical</td>
<td>Relationship</td>
<td>The relationship between an Annotation, Body or Target and the IRI that should be</td>
</tr>
</tbody>
</table>
used to track its identity, regardless of where it is made accessible. If this property is set, then systems must not change or delete it. Systems should not assign a canonical IRI without prior agreement if one is not present, as the Annotation could already have a canonical IRI elsewhere.

There may be exactly 1 canonical IRI for each resource.

| via | Relationship | The relationship between an Annotation, Body or Target and the IRI of where that resource was obtained from by the system that is making it available. | There may be 0 or more IRIs provided in via for each resource. |

**Example**

**EXAMPLE 17: Other Identities**

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno17",
  "type": "Annotation",
  "canonical": "urn:uuid:dbfb1861-0ecf-41ad-be94-a584e5c4f1df",
  "via": "http://other.example.org/anno1",
  "body": {
    "id": "http://example.net/review1",
    "rights": "http://creativecommons.org/licenses/by/4.0/"
  },
  "target": "http://example.com/product1"
}
```

4. Specific Resources

While it is possible using only the constructions described above to create Annotations that reference parts of resources by using IRIs with a fragment component, there are many situations when this is not sufficient. For example, even a simple circular region of an image, or a diagonal line across it, are not possible. Selecting an arbitrary span of text in an HTML page, perhaps the simplest annotation concept, is also not supported by fragments. Furthermore, there are non-segment use cases that require a client to retrieve a specific state or representation of the resource, to style it in a particular way, to associate a role with the resource that is specific to the Annotation's use of it, or for the Annotation to only apply when the resource is used in a particular context.

The Web Annotation Data Model uses a new type of resource to capture these Annotation-specific requirements: a SpecificResource. The SpecificResource is used in between the Annotation and the Body or Target, as appropriate, to capture additional information about how it is used in the Annotation. The descriptions are typically referenced from the SpecificResource as separate entities and can be of various types to capture the different requirements. For example, if the Target of the Annotation is a circular region of an image, then the SpecificResource is the circular region, it is described by a Selector, and is also associated with the source Image resource.

Specific Resources and Specifiers may be External Web Resources with their own IRIs, such as in the example for the **Selector** construction, however it is recommended that they be included in the Annotation's representation to avoid requiring un-
necessary network interactions to retrieve all of the information needed to process the Annotation.

The types of additional specificity that are defined by this document:

- **Purpose**: Describe the purpose of including the source resource in the Annotation
- **Selector**: Describe the desired segment of the source resource for the Annotation
- **State**: Describe the desired representation of the source resource for the Annotation
- **Style**: Describe the style in which the source resource should be presented for the Annotation
- **Rendering**: Describe the system used by the client for rendering the resource when the annotation was created
- **Scope**: Describe the scope in which the source resource applies for the Annotation

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The identity of the Specific Resource. A Specific Resource may have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Specific Resource. The Specific Resource should have the SpecificResource class.</td>
</tr>
<tr>
<td>SpecificResource</td>
<td>Class</td>
<td>The class for Specific Resources. The SpecificResource class should be associated with a Specific Resource to be clear as to its role as a more specific region or state of another resource.</td>
</tr>
<tr>
<td>source</td>
<td>Relationship</td>
<td>The relationship between a Specific Resource and the resource that it is a more specific representation of. There must be exactly 1 source relationship associated with a Specific Resource. The source resource may be described in detail, as defined above, or be just the resource's IRI.</td>
</tr>
</tbody>
</table>

The same Specific Resource and Specifier classes are used for both Target and Body. The examples in this section only use one of these, however the same model applies for both.

### 4.1 Purpose for External Web Resources

As well as Textual Bodies, External Web Resources may also be given a Motivation as to their inclusion within the Annotation. This is done using the Specific Resource pattern, as the purpose specifies the way in which the resource is used in the context of the Annotation in the same way as a Selector describes the segment or a State describes the representation.

**Example Use Case**: Qitara wants to tag a photo with an identifier for a city, rather than just type the city's name which could be ambiguous. Her client uses a well-known IRI for the city having done a search for it, and creates a Specific Resource to manage that purpose assignment.

### Model
<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>Relationship</td>
<td>The reason for including the Web Resource in the Annotation. There may be 0 or more Motivations associated with the SpecificResource using purpose.</td>
</tr>
</tbody>
</table>

**Example**

**EXAMPLE 18: Resource with Purpose**

```
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno18",
    "type": "Annotation",
    "body": {
        "type": "SpecificResource",
        "purpose": "tagging",
        "source": "http://example.org/city1"
    },
    "target": {
        "id": "http://example.org/photo1",
        "type": "Image"
    }
}
```

**4.2 Selectors**

Many Annotations refer to part of a resource, rather than all of it, as the Target. We call that part of the resource a Segment (of Interest). A Selector is used to describe how to determine the Segment from within the Source resource. The nature of the Selector will be dependent on the type of resource, as the methods to describe Segments from various media-types will differ. Multiple Selectors can be given to describe the same Segment in different ways in order to maximize the chances that it will be discoverable later, and that the consuming user agent will be able to use at least one of the Selectors.

**Example Use Case:** Ramona wants to associate a selection of text in a web page, with a slice of a dataset. She selects both using her client, and creates the Annotation with a SpecificResource that has a Selector for each of the Body and the Target.

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>Relationship</td>
<td>The relationship between a Specific Resource and a Selector. There may be 0 or more selector relationships associated with a Specific Resource. Multiple Selectors should select the same content, however some Selectors will not have the same precision as others. Consuming user agents must pick one of the described segments, if they are different.</td>
</tr>
</tbody>
</table>
4.2.1 Fragment Selector

As the most well understood mechanism for selecting a Segment is to use the fragment part of an IRI defined by the representation's media type, it is useful to allow this as a description mechanism via a Selector. This allows existing and future fragment specifications to be used with Specific Resources in a consistent way. To be clear about which fragment type is being used, the Selector may refer to the specification that defines it.

**Example Use Case:** Sally wants to associate part of a video as the description of an image. She selects the time range within the video and clicks that it is describing the target. Her client then creates the Annotation using a SpecificResource with a `FragmentSelector` and the `describing` Motivation.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>FragmentSelectors must have exactly 1 <code>type</code> and the value must be <code>FragmentSelector</code>.</td>
</tr>
<tr>
<td>FragmentSelector</td>
<td>Class</td>
<td>A resource which describes the Segment through the use of the fragment component of an IRI.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The contents of the fragment component of an IRI that describes the Segment. The FragmentSelector must have exactly 1 <code>value</code> property.</td>
</tr>
<tr>
<td>conformsTo</td>
<td>Relationship</td>
<td>The relationship between the FragmentSelector and the specification that defines the syntax of the IRI fragment in the <code>value</code> property. The Fragment Selector should have exactly 1 <code>conformsTo</code> link to the</td>
</tr>
</tbody>
</table>
It is recommended to use `FragmentSelector` as a consistent method compatible with other means of describing Specific-Resources, rather than using the IRI with a fragment directly. Consuming applications should be aware of both.

The following IRIs are some of the specifications that define the semantics of fragments, and hence may be used with the `conformsTo` relationship. Other IRIs may also be used.

<table>
<thead>
<tr>
<th>Name</th>
<th>Fragment Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain Text</td>
<td><a href="http://tools.ietf.org/rfc/rfc5147">http://tools.ietf.org/rfc/rfc5147</a></td>
<td>[rfc5147] Example: <code>char=0,10</code></td>
</tr>
<tr>
<td>Media</td>
<td><a href="http://www.w3.org/TR/media-frags/">http://www.w3.org/TR/media-frags/</a></td>
<td>[media-frags] Example: <code>xywh=50,50,640,480</code></td>
</tr>
<tr>
<td>SVG</td>
<td><a href="http://www.w3.org/TR/SVG/">http://www.w3.org/TR/SVG/</a></td>
<td>[SVG11] Example: <code>svgView(viewBox(50,50,640,480))</code></td>
</tr>
</tbody>
</table>

**NOTE**

The IRI that uses the fragment may be reconstructed by concatenating the `source`, a `#`, and the `value`. For example, the IRI from the example below would be `http://example.org/video1#t=30,60`.

**Example**
4.2.2 CSS Selector

One of the most common ways to select elements in the HTML Document Object Model is to use CSS Selectors [CSS3-selectors]. CSS Selectors allow for a wide variety of well supported ways to describe the path to an element in a web page, and thus cover many of the basic use cases for Web Annotation. Results are not defined for when a CSS Selector is applied to a representation that does not conform to the Document Object Model.

Note that CSS may also be used for styling a resource within an annotation. This class is specifically to re-use the CSS Selector mechanism to select a segment of a resource that conforms to the Document Object Model.

**Example Use Case:** Teynika selects a paragraph in a web page that she wishes to write a note about. Her client calculates a CSS path that cleanly identifies that element and adds it to the annotation.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Selector. CssSelectors must have exactly 1 type and the value must be CssSelector.</td>
</tr>
<tr>
<td>CssSelector</td>
<td>Class</td>
<td>The type of the CSS Selector resource. CSS Selectors must have this class associated with them.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The CSS selection path to the Segment. There must be exactly 1 value associated with a CSS Selector.</td>
</tr>
</tbody>
</table>

**NOTE**

Implementers should use only commonly supported features of CSS that directly contribute to selection of an element or content, rather than styling or transformation, in order to maximize interoperability between systems.
4.2.3 XPath Selector

Another common method of selecting elements and content within a resource that supports the Document Object Model (DOM), such as documents in XML or HTML, is to use an XPath selection [DOM-Level-3-XPath]. XPath allows a great deal of flexibility when describing the path through the structure to the selected content. Results are not defined for when an XPath Selector is applied to a representation that does not conform to the DOM.

NOTE
Implementers should note that the HTML5 specification allows parsers to add elements into the DOM that are considered to be missing. XPaths should be constructed to include these elements, rather than from the element structure in the document.

Example Use Case: Ulrika selects a span within a table in an HTML page and writes a note about the content. To refer explicitly to this element, her client carefully constructs an XPath to identify it as the target of the Annotation.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relation</td>
<td>The class of the Selector. XPath Selectors must have exactly 1 type and the value must be XPathSelector.</td>
</tr>
<tr>
<td>XPathSelector</td>
<td>Class</td>
<td>The type of the XPath Selector resource. XPath Selectors must have this class associated with them.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The xpath to the selected segment. There must be exactly 1 value associated with an XPath Selector.</td>
</tr>
</tbody>
</table>
4.2.4 Text Quote Selector

This Selector describes a range of text by copying it, and including some of the text immediately before (a prefix) and after (a suffix) it to distinguish between multiple copies of the same sequence of characters.

For example, if the document were again "abcdefgijklmnopqrstuvwxyz", one could select "efg" by a prefix of "abcd", the match of "efg" and a suffix of "hijk".

**Example Use Case:** Valeria selects a typo ('anotation') in a web page and adds a comment that it should be replaced with the correct spelling ('annotation').

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Selector. Text Quote Selectors must have exactly 1 type and the value must be TextQuoteSelector.</td>
</tr>
<tr>
<td>TextQuoteSelector</td>
<td>Class</td>
<td>The class for a Selector that describes a textual segment by means of quoting it, plus passages before or after it. The TextQuoteSelector must have this class associated with it.</td>
</tr>
<tr>
<td>exact</td>
<td>Property</td>
<td>A copy of the text which is being selected, after normalization.</td>
</tr>
</tbody>
</table>

**NOTE**
Implementers should use only commonly supported features of XPath that directly contribute to selection of an element or content in order to maximize interoperability between systems.
Each TextQuoteSelector must have exactly 1 `exact` property.

| prefix | Property | A snippet of text that occurs immediately before the text which is being selected. Each TextQuoteSelector should have exactly 1 `prefix` property, and must not have more than 1. |
| suffix | Property | The snippet of text that occurs immediately after the text which is being selected. Each TextQuoteSelector should have exactly 1 `suffix` property, and must not have more than 1. |

The selection of the text must be in terms of unicode code points (the "character number"), not in terms of code units (that number expressed using a selected data type). Selections should not start or end in the middle of a grapheme cluster. The selection must be based on the logical order of the text, rather than the visual order, especially for bidirectional text. For more information about the character model of text used on the web, see [charmod].

The text must be normalized before recording in the Annotation. Thus HTML/XML tags should be removed, and character entities should be replaced with the character that they encode. Note that this does not affect the state of the content of the document being annotated, only the way that the content is recorded in the Annotation document.

If, after processing the prefix, exact, and suffix, the user agent discovers multiple matching text sequences, then the selection should be treated as matching all of the matches.

**NOTE**

If the content is under copyright or has other rights asserted on its use, then this method of selecting text is potentially dangerous. A user might select the entire text of the document to annotate, which would not be desirable to copy into the Annotation and share. For static texts with access and/or distribution restrictions, the use of the Text Position Selector is perhaps more appropriate.

**Example**
4.2.5 Text Position Selector

This Selector describes a range of text by recording the start and end positions of the selection in the stream. Position 0 would be immediately before the first character, position 1 would be immediately before the second character, and so on. The start character is thus included in the list, but the end character is not.

For example, if the document was "abcdefghijklmnopqrstuvwxyz", the start was 4, and the end was 7, then the selection would be "efg".

Example Use Case: Wendy writes a review of an ebook that does not allow its content to be extracted and copied. Her client describes the selection using its start and end position in the content.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Selector. Text Position Selectors <strong>must</strong> have exactly 1 type and the value <strong>must</strong> be TextPositionSelector.</td>
</tr>
<tr>
<td>TextPositionSelector</td>
<td>Class</td>
<td>The class for a Selector which describes a range of text based on its start and end positions. The TextPositionSelector <strong>must</strong> have this class associated with it.</td>
</tr>
<tr>
<td>start</td>
<td>Property</td>
<td>The starting position of the segment of text. The first character in the full text is character position 0, and the character is included within the segment. Each TextPositionSelector <strong>must</strong> have exactly 1 start property, and the value <strong>must</strong> be a non-negative integer.</td>
</tr>
<tr>
<td>end</td>
<td>Property</td>
<td>The end position of the segment of text. The character is not included.</td>
</tr>
</tbody>
</table>
within the segment. Each TextPositionSelector must have exactly 1 end property, and the value must be a non-negative integer.

The text must be selected and normalized in the same way as for the Text Quote Selector before counting the number of characters to determine the start and end positions.

NOTE
The use of this Selector does not require text to be copied from the Source document into the Annotation graph, unlike the Text Quote Selector, but is very brittle with regards to changes to the resource. Any edits or dynamically transcluded content may change the selection, and thus it is recommended that a State be additionally used to help identify the correct representation.

Example

EXAMPLE 24: Text Position Selector

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno24",
    "type": "Annotation",
    "body": "http://example.org/review1",
    "target": {
        "source": "http://example.org/ebook1",
        "selector": {
            "type": "TextPositionSelector",
            "start": 412,
            "end": 795
        }
    }
}
```

4.2.6 Data Position Selector

Similar to the Text Position Selector, the Data Position Selector uses the same properties but works at the byte in bitstream level rather than the character in text level.

Example Use Case: Xena writes comments about regions of online disk images for forensic purposes and describing emulation requirements. Her client generates the start and end positions from the binary stream, rather than the more human readable display she is using.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>type</strong></td>
<td><strong>Relationship</strong></td>
<td><strong>Data Position Selectors</strong> must have exactly 1 <strong>type</strong> and the value <strong>must</strong> be <strong>DataPositionSelector</strong>.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>DataPositionSelector</strong></td>
<td><strong>Class</strong></td>
<td>The class for a Selector which describes a range of data based on its start and end positions within the byte stream. The DataPositionSelector must have this class associated with it.</td>
</tr>
<tr>
<td><strong>start</strong></td>
<td><strong>Property</strong></td>
<td>The starting position of the segment of data. The first byte is character position 0. Each DataPositionSelector must have exactly 1 <strong>start</strong> property.</td>
</tr>
<tr>
<td><strong>end</strong></td>
<td><strong>Property</strong></td>
<td>The end position of the segment of data. The last character is not included within the segment. Each DataPositionSelector must have exactly 1 <strong>end</strong> property.</td>
</tr>
</tbody>
</table>

**Example**

**EXAMPLE 25: Data Position Selector**

```
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno25",
    "type": "Annotation",
    "body": "http://example.org/note1",
    "target": {
        "source": "http://example.org/diskimg1",
        "selector": {
            "type": "DataPositionSelector",
            "start": 4096,
            "end": 4104
        }
    }
}
```

**4.2.7 SVG Selector**

An SvgSelector defines an area through the use of the Scalable Vector Graphics [SVG11] standard. This allows the user to select a non-rectangular area of the content, such as a circle or polygon by describing the region using SVG. The SVG may be either embedded within the Annotation or referenced as an External Web Resource.

Note that the SvgSelector uses SVG to select an area of a resource. Segments of an SVG representation may also be selected using selectors, including the FragmentSelector or even an SvgSelector.

**Example Use Case:** Yadira is tagging an old map online with a diagonal region for a historical road. Her client creates SVG polygon to describe the region, relative to the image content.
### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Selector. SVG Selectors <strong>must</strong> have exactly 1 <code>type</code> and the value <strong>must</strong> include <code>SvgSelector</code>.</td>
</tr>
<tr>
<td>SvgSelector</td>
<td>Class</td>
<td>The class for a Selector which defines a shape for the selected area using the SVG standard. The Selector <strong>must</strong> have this class associated with it.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The character sequence of the SVG content. There <strong>may</strong> be exactly 1 <code>value</code> property associated with the Selector, and if so the value of the property <strong>must</strong> be well-formed SVG XML.</td>
</tr>
</tbody>
</table>

The dimensions of the SVG shape or canvas **must** be relative to the dimensions of the Source resource, such that scaling the shape's size to the full size of the image correctly describes the desired area.

**NOTE**

Implementers should use only commonly supported features of SVG that directly contribute to describing a region, rather than styling or transformation, in order to maximize interoperability between systems. It is not recommended to include style information within the SVG element, nor Javascript, animation, text or other non-shape oriented information. Clients should ignore such information if present.

### Example

**EXAMPLE 26: SVG Selector**

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno26",
  "type": "Annotation",
  "body": "http://example.org/road1",
  "target": {
    "source": "http://example.org/map1",
    "selector": {
      "id": "http://example.org/svg1",
      "type": "SvgSelector"
    }
  }
}
```
4.2.8 Range Selector

Selections made by users may be extensive and/or cross over internal boundaries in the representation, making it difficult to construct a single selector that robustly describes the correct content. A Range Selector can be used to identify the beginning and the end of the selection by using other Selectors. In this way, two points can be accurately identified using the most appropriate selection mechanisms, and then linked together to form the selection. The selection consists of everything from the beginning of the starting selector through to the beginning of the ending selector, but not including it.

Example Use Case: Zara wants to comment on two adjacent cells in a table that is part of a web page. She selects the two cells and her client constructs XPaths to the the first cell, and the cell that immediately follows the second. Her client then creates a Range Selector with the first XPath Selector as the start, and the second XPath selector as the end.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Selector. Range Selectors must have exactly 1 type and the value must be RangeSelector.</td>
</tr>
<tr>
<td>RangeSelector</td>
<td>Class</td>
<td>The type of the Range Selector resource. Range Selectors must have this class associated with them.</td>
</tr>
<tr>
<td>startSelector</td>
<td>Relationship</td>
<td>The Selector which describes the inclusive starting point of the range. There must be exactly 1 startSelector associated with a Range Selector.</td>
</tr>
<tr>
<td>endSelector</td>
<td>Relationship</td>
<td>The Selector which describes the exclusive ending point of the range. There must be exactly 1 endSelector associated with a Range Selector. Both startSelector and endSelector should be of the same class.</td>
</tr>
</tbody>
</table>

Example
4.2.9 Refinement of Selection

It may be easier, more reliable or more accurate to specify the segment of interest of a resource as a selection of a selection, rather than as a selection of the complete resource. Particularly for resources that contain other resources, such as various packaging formats, this also allows decomposition of the selection mechanisms when the components do not have unique identifiers. This is accomplished by having selectors chained together, where each refines the results of the previous one.

Example Use Case: Alexandra selects a paragraph of text and then a short phrase within it to comment on. Her client records the phrase as a TextQuoteSelector that further modifies a FragmentSelector used to identify the paragraph that the phrase is part of.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refinedBy</td>
<td>Relationship</td>
<td>The relationship between a broader selector and the more specific selector that should be applied to the results of the first. A Selector may be refinedBy 1 or more other Selectors. If more than 1 is given, then they are considered to be alternatives that will result in the same selection.</td>
</tr>
</tbody>
</table>

Example

```json

EXAMPLE 28: Range Selector

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno28",
  "type": "Annotation",
  "body": "http://example.org/comment1",
  "target": {
    "source": "http://example.org/page1.html",
    "selector": {
      "type": "RangeSelector",
      "startSelector": {
        "type": "XPathSelector",
        "value": "/table[1]/tr[1]/td[2]"
      },
      "endSelector": {
        "type": "XPathSelector",
        "value": "/table[1]/tr[1]/td[4]"
      }
    }
  }
}
```
4.3 States

A State describes the intended state of a resource as applied to the particular Annotation, and thus provides the information needed to retrieve the correct representation of that resource. Web resources change over time, and a State might be used to describe how to recover the intended previous version. Web resources also have multiple formats, and a State might equally be used to describe how to retrieve that particular format. Multiple States may be given to describe the same representation in order to maximize the chances that the representation will be retrievable by the consuming user agent.

**Example Use Case:** Britney makes a comment about a web page that changes frequently. Her client records information to allow other clients to hopefully reconstruct the original target of the annotation.

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>Relationship</td>
<td>The relationship between the SpecificResource and the State. There may be 0 or more state relationships for each SpecificResource. Multiple States should describe the same representation, however some States will not have the same precision as others. Consuming user agents must pick one of the described representations, if they are different.</td>
</tr>
</tbody>
</table>

States must be processed before processing Selector or Style information.

**Example**
### 4.3.1 Time State

A Time State resource records the time at which the resource is appropriate for the Annotation, typically the time that the Annotation was created and/or a link to a persistent copy of the current version. The timestamp for the resource could be resolved via the Memento protocol, described in RFC 7089 [rfc7089].

**Example Use Case:** Carla makes a note about the current state of the front page of a news website, and flags that the page is likely to change often. Her client adds in a State with the current time to describe the version of the page being annotated.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the State. Time States must have exactly 1 type and the value must be TimeState.</td>
</tr>
<tr>
<td>TimeState</td>
<td>Class</td>
<td>A description of how to retrieve a representation of the Source resource that is temporally appropriate for the Annotation. The State must have this class associated with it.</td>
</tr>
<tr>
<td>sourceDate</td>
<td>Property</td>
<td>The timestamp at which the Source resource should be interpreted for the Annotation. There may be 0 or more sourceDate properties per TimeState. If there is more than 1, each gives an alternative timestamp at which the Source may be interpreted. The timestamp must be expressed in the xsd:dateTime format, and must use the UTC timezone expressed as &quot;Z&quot;. If sourceDate is provided, then sourceDateStart and sourceDateEnd must not be provided.</td>
</tr>
<tr>
<td>sourceDateStart</td>
<td>Property</td>
<td>The timestamp that begins the interval over which the Source resource should be interpreted for the Annotation. There may be exactly 1 sourceDateStart property per TimeState. The timestamp must be expressed in the xsd:dateTime format, and must use the</td>
</tr>
</tbody>
</table>
UTC timezone expressed as “Z”. If `sourceDateStart` is provided then `sourceDateEnd` must also be provided.

| sourceDateEnd | Property | The timestamp that ends the interval over which the Source resource should be interpreted for the Annotation. There may be exactly 1 `sourceDateEnd` property per TimeState. The timestamp must be expressed in the `xsd:dateTime` format, and must use the UTC timezone expressed as “Z”. If `sourceDateEnd` is provided then `sourceDateStart` must also be provided. |
| cached | Relationship | A link to a copy of the Source resource’s representation, appropriate for the Annotation. There may be 0 or more `cached` relationships per TimeState. If there is more than 1, each gives an alternative copy of the representation. |

**Example**

EXAMPLE 31: Time State

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno31",
  "type": "Annotation",
  "body": "http://example.org/note1",
  "target": {
    "source": "http://example.org/page1",
    "state": {
      "type": "TimeState",
      "cached": "http://archive.example.org/copy1",
      "sourceDate": "2015-07-20T13:30:00Z"
    }
  }
}
```

**4.3.2 Request Header State**

As there are potentially many representations that can be delivered from a resource with a single IRI, and a Specific Resource may only apply to one of them, it is important to be able to record the HTTP Request headers that need to be sent to retrieve the correct representation. The HttpRequestState resource maintains a copy of the headers to be replayed when obtaining the representation.

**Example Use Case:** Devina retrieves a PDF representation of a web resource that can deliver HTML, PDF or plain text and then writes a description about it. She signals that her description is only about the PDF representation. Her client then includes a State to describe how to retrieve the target representation.

**Model**
<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the State. Request Header States must have exactly 1 <code>type</code> and the value must be <code>HttpRequestState</code>.</td>
</tr>
<tr>
<td>HttpRequestState</td>
<td>Class</td>
<td>A description of how to retrieve an appropriate representation of the Source resource for the Annotation, based on the HTTP Request headers to send on the request. The State must have this class associated with it.</td>
</tr>
<tr>
<td>value</td>
<td>Property</td>
<td>The HTTP request headers to send as a single, complete string. An HttpRequestState must have exactly 1 <code>value</code> property.</td>
</tr>
</tbody>
</table>

**NOTE**

The representation retrieved from the server by the original annotator's client might not be completely determined by request headers alone. For example, the IP address of the client might also determine the language of the representation, based on the language of the country the user was present in at the time. If the server returns a `Content-Location` header, then the client might instead use it as the `target` of the Annotation, rather than the IRI that was requested.

**Example**

**EXAMPLE 32: HTTP Request State**

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno32",
  "type": "Annotation",
  "body": "http://example.org/description1",
  "target": {
    "source": "http://example.org/resource1",
    "state": {
      "type": "HttpRequestState",
      "value": "Accept: application/pdf"
    }
  }
}
```

**4.3.3 Refinement of State**

Similar to the refinement of selection, it may be easier, more reliable or more accurate to specify the appropriate state of the resource as a hierarchy of atomic State resources. This is particularly appropriate for representing the combination of a State that reflects an internal transformation along with the results of a State that describes an external request. This decomposition is accomplished by having the states chained together in the same way as Selectors.
Further, given that the State(s) will likely result in a specific representation, there may be specific Selectors that are appropriate for describing the segment of the representation. In order to accommodate this, States may also be refined by Selectors.

**Example Use Case:** Erin writes a comment about a travel e-book which has many versions available over time, and is available in different formats. She is particularly commenting on a specific version and format, so her client adds both a TimeState to capture the time and an HttpRequestState to capture the format, and then a particular FragmentSelector that is appropriate to that format.

### Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>refinedBy</td>
<td>Relationship</td>
<td>The relationship between a broader State and either a more specific State or a Selector that <strong>should</strong> be applied to the results of the first. Each State <strong>may be</strong> refinedBy 1 or more other States or Selectors. If more than 1 is given, then they are considered to be alternatives that will result in the same result.</td>
</tr>
</tbody>
</table>

### Example

**EXAMPLE 33: Refinement of States**

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno33",
    "type": "Annotation",
    "body": "http://example.org/comment1",
    "target": {
        "source": "http://example.org/ebook1",
        "state": {
            "type": "TimeState",
            "sourceDate": "2016-02-01T12:05:23Z",
            "refinedBy": {
                "type": "HttpRequestState",
                "value": "Accept: application/pdf",
                "refinedBy": {
                    "type": "FragmentSelector",
                    "value": "page=10",
                    "conformsTo": "http://tools.ietf.org/rfc/rfc3778"
                }
            }
        }
    }
}
```

### 4.4 Styles
The interpretation of a particular Annotation, or the Annotation's Body, may rely on the rendering style of the Annotation being consistent across implementations. For Annotations on binary content such as Images or Video, the background color of the Target may not be accessible to the annotation client, and the default coloring may be difficult to perceive, such as a black rectangle rendered as the target area on an image of the night sky. Rendering information is recorded using CSS stylesheets and references to classes defined in those stylesheets.

**Example Use Case:** Felicity highlights two paragraphs in a document, and selects in her client that one should be highlighted in red and the other in yellow. She then makes a comment that the yellow part contradicts the red part. Her client records that she selected the red and yellow coloration of the targets.

---

## Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The class of the Style. CSS Stylesheets may have a type and if included the value must be.CssStylesheet.</td>
</tr>
<tr>
<td>CssStylesheet</td>
<td>Class</td>
<td>A resource which describes styles for resources participating in the Annotation using CSS. The class may be associated with the stylesheet resource.</td>
</tr>
<tr>
<td>stylesheet</td>
<td>Relationship</td>
<td>The relationship between an Annotation and the Style. There may be 0 or 1 stylesheet relationships for each Annotation.</td>
</tr>
<tr>
<td>styleClass</td>
<td>Property</td>
<td>The name of the class used in the CSS description that should be applied to the Specific Resource. There may be 0 or more styleClass properties on a Specific Resource.</td>
</tr>
</tbody>
</table>

The CSS Stylesheet is associated with the Annotation itself, and the content provides the rendering hints about the Annotation's constituent resources. It may have its own dereferenceable IRI that provides the information, or it may be embedded within the Annotation. This is to avoid having single line style sheets each associated with different resources, and instead to allow reference to a single IRI that governs the full set of styles for a particular implementation.

Publishing systems must not assume that they will be processed; they are only provided as hints rather than requirements.

When rendering a Specific Resource, consuming applications should check to see if it has a styleClass property. If it does, then the application should attempt to locate the appropriate selector in the CSS document, and then apply the css-value block. If a Specific Resource has a styleClass value, but no such class is described by a stylesheet attached to the Annotation, then the styleClass must be ignored.

---

## Example
4.5 Rendering Software

It may be valuable to know the software that was used to process and/or render the Target resource when the annotation was created. This information can be used by later systems to potentially recreate the environment to ensure that the annotation can be more easily and more accurately reconnected with the appropriate part of the Target's representation. This life cycle information is associated with the Specific Resource, as it is very likely to change between Annotations for the same Target, and thus cannot be associated with the Target resource directly.

Example Use Case: Gabrielle is using a browser based client to render a PDF of a scholarly article. Her browser uses a particular library to render the PDF as HTML. She annotates a paragraph in the view that she sees, the HTML rendering, and her client records that the library that was used for rendering in the annotation, along with her comment and the target PDF.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMPLE 34: CSS Style</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>EXAMPLE 35: CSS Style, embedded</td>
<td>{</td>
<td></td>
</tr>
</tbody>
</table>

renderedVia Relationship The relationship between the Specific Resource that represents the Target in the annotation, and the piece of software or other system that was used to render the Target when the annotation was created. There may be 0 or more renderedVia relationships for each Specific Resource.

NOTE
Other properties may be associated with the rendering system, including such things as schema:softwareVersion, accessibility functions, labels, references to the actual code, and so forth. These extensions are beyond the scope of this specification, but please see the Extensions section of [annotation-vocab].

Example

EXAMPLE 36: Rendering Software

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno36",
  "type": "Annotation",
  "body": "http://example.org/comment1",
  "target": {
    "source": "http://example.edu/article.pdf",
    "selector": "http://example.org/selectors/html-selector1",
    "renderedVia": {
      "id": "http://example.com/pdf-to-html-library",
      "type": "Software",
      "schema:softwareVersion": "2.5"
    }
  }
}
```

4.6 Scope of a Resource

It is sometimes important for an Annotation to capture the context in which it was made, in terms of the resources that the annotator was viewing or using at the time. This does not imply an assertion that the annotation is only valid for the image in the context of that page, it just records that the page was being viewed.

Example Use Case: Heather makes a comment about an image in a particular web page to say that it is not the right organization's logo. Her client includes the page that the image is being rendered in, however the annotation is associated with the image resource itself.

Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>Relationship</td>
<td>The relationship between a Specific Resource and the resource that provides the scope</td>
</tr>
</tbody>
</table>
5. Collections

It is often useful to be able to collect Annotations together into a list, called an Annotation Collection. This list, which is always ordered, serves as a means to refer to the Annotations that are contained within it, and to maintain any information about the Collection itself.

The Collection model is divided into two sections: the Annotation Collection that manages the identity of the list and its description, and Annotation Pages that list the Annotations which are members of the Collection.

Example Use Case: Ingeborg works for a publishing house and has transformed the author’s commentary on their steampunk novel into a set of annotations for sale. The company wishes to have them available as an add-on for customers that have already bought the novel, and also in a bundle for new sales.

5.1 Annotation Collection

As Annotation Collections might get very large, the model distinguishes between the Collection itself and sequence of component pages that in turn list the Annotations. The Collection maintains information about itself, including creation or descriptive information to aid with discovery and understanding of the Collection, and also references to at least the first Page of Annotations. By starting with the first Annotation in the first Page, and traversing the Pages to the last Annotation of the last Page, all Annotations in the Collection will have been discovered.

Annotations may be within multiple Collections at the same time, and the Collection may be created or maintained by agents other than those that create or maintain the included Annotations.
<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td>Property</td>
<td>The context that determines the meaning of the JSON as an Annotation Collection. The Collection must have 1 or more @context values and <a href="http://www.w3.org/ns/anno.jsonld">http://www.w3.org/ns/anno.jsonld</a> must be one of them.</td>
</tr>
<tr>
<td>id</td>
<td>Property</td>
<td>The identity of the Collection. The Collection must have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Property</td>
<td>The type of the Collection. The Collection must have 1 or more types, and the AnnotationCollection class must be one of them.</td>
</tr>
<tr>
<td>AnnotationCollection</td>
<td>Class</td>
<td>The class for ordered Collections ofAnnotations. This class must be associated with the Collection using type.</td>
</tr>
<tr>
<td>label</td>
<td>Property</td>
<td>A human readable label intended as the name of the Collection. Collections should have 1 or more labels, and the value must be a string.</td>
</tr>
<tr>
<td>total</td>
<td>Property</td>
<td>The total number of Annotations in the Collection. Collections should have exactly 1 total, and if present it must be an xsd:nonNegativeInteger.</td>
</tr>
<tr>
<td>first</td>
<td>Relationship</td>
<td>The first page of Annotations that are included within the Collection. A Collection that has a total number of Annotations greater than 0 must have exactly 1 first page of Annotations. The first Page may be embedded within the representation of the Collection, or it may be given as an IRI.</td>
</tr>
<tr>
<td>last</td>
<td>Relationship</td>
<td>The last page of Annotations that are included within the Collection. A Collection that has a total number of Annotations greater than 0 should have a reference to the IRI of the last page of Annotations.</td>
</tr>
</tbody>
</table>

Other properties may be added to the Collection to describe its use, intellectual property rights, provenance and any other features that are considered useful. These properties should come from those described in this specification if possible, but may come from any appropriate vocabulary.

**Example**
5.2 Annotation Page

An Annotation Page is part of an Annotation Collection, and has an ordered list of some or all of the Annotations that are within the Collection. Each Collection may have multiple pages, and these are traversed by following the next and prev links between the pages.

**Model**

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@context</td>
<td>Property</td>
<td>The context that determines the meaning of the JSON as an Annotation Collection Page. If the Page is NOT embedded within a Collection, it must have 1 or more @context values and <a href="http://www.w3.org/ns/anno.jsonld">http://www.w3.org/ns/anno.jsonld</a> must be one of them. If it is embedded, then it should not have an @context property.</td>
</tr>
<tr>
<td>id</td>
<td>Property</td>
<td>The identity of the Page. The Page must have exactly 1 IRI that provides its identity.</td>
</tr>
<tr>
<td>type</td>
<td>Property</td>
<td>The type of the Page. The Page must have 1 or more types, and the AnnotationPage class must be one of them.</td>
</tr>
<tr>
<td>AnnotationPage</td>
<td>Class</td>
<td>The class of Annotation Pages. This class must be associated with the Page using type.</td>
</tr>
<tr>
<td>partOf</td>
<td>Relationship</td>
<td>The relationship between the Page and the Annotation Collection that it is part of. Each Page should have a exactly 1 partOf relationship, with the value being either the IRI of the Collection or an object with some or all of the Collections properties, including at least its id.</td>
</tr>
<tr>
<td>items</td>
<td>Relationship</td>
<td>The list of Annotations that are the members of the Page. Each Page must have an array of 1 or more Annotations as the value of items.</td>
</tr>
<tr>
<td>next</td>
<td>Relationship</td>
<td>A reference to the next Page in the sequence of pages that make up the</td>
</tr>
</tbody>
</table>
Collection.
If the current page is not the last page in the Collection, it must have a reference to the IRI of the page that follows it.

prev Relationship
A reference to the previous Page in the sequence of pages that make up the Collection.
If the current page is not the first page in the Collection, it should have a reference to the IRI of the page that follows.

startIndex Property
The relative position of the first Annotation in the items list, relative to the Annotation Collection. The first entry in the first page is considered to be entry 0.
Each Page should have exactly 1 startIndex, and must not have more than 1. The value must be an xsd:nonNegativeInteger.

Example

EXAMPLE 39: Annotation Page

```json
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/page1",
  "type": "AnnotationPage",
  "partOf": {
    "id": "http://example.org/collection1",
    "label": "Steampunk Annotations",
    "total": 42023
  },
  "next": "http://example.org/page2",
  "startIndex": 0,
  "items": [
    {
      "id": "http://example.org/anno1",
      "type": "Annotation",
      "body": "http://example.net/comment1",
      "target": "http://example.com/book/chapter1"
    },
    {
      "id": "http://example.org/anno2",
      "type": "Annotation",
      "body": "http://example.net/comment2",
      "target": "http://example.com/book/chapter2"
    }
  ]
}
```
EXAMPLE 40: Annotation Collection with Embedded Page

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/collection1",
  "type": "AnnotationCollection",
  "label": "Two Annotations",
  "total": 2,
  "first": {
    "id": "http://example.org/page1",
    "type": "AnnotationPage",
    "startIndex": 0,
    "items": [
      {
        "id": "http://example.org/anno1",
        "type": "Annotation",
        "body": "http://example.net/comment1",
        "target": "http://example.com/book/chapter1"
      },
      {
        "id": "http://example.org/anno2",
        "type": "Annotation",
        "body": "http://example.net/comment2",
        "target": "http://example.com/book/chapter2"
      }
    ]
  }
}
```

A. Correspondence Among Media Types and Selectors

The table below shows the relationships among major media types and selector types. It is relevant to the 1.3 Conformance section of this document.

<table>
<thead>
<tr>
<th>Media Type</th>
<th>Fragment</th>
<th>CSS</th>
<th>XPath</th>
<th>Text Quote</th>
<th>Text Position</th>
<th>Data Position</th>
<th>Svg</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML (text/html)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>CSV (text/csv)</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Plain Text (text/plain)</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Other text files (text/*)</td>
<td>?</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>EPUB2, EPUB3 (application/epub+zip)</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>PDF (application/pdf)</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>XML (application/xml, application/*+xml)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>SVG (image/svg+xml)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
</tr>
</tbody>
</table>
A.1 Additional Media Types/Selector Combination

This section is non-normative.

The table below contains some other, possible combinations of media types and selector types, which may be implemented but are not mandated by this specification. Some of these combinations may also form the basis for defining new, implementation-specific selector extensions.

<table>
<thead>
<tr>
<th>Fragment CSS XPath Text Quote Text Position Data Position Svg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image, other than SVG (image/gif, image/jpeg, image/png, image/tiff)</td>
</tr>
<tr>
<td>Video (video/*)</td>
</tr>
<tr>
<td>Binary Data Files</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional relationships among other media types and selector types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS (text/css)</td>
</tr>
<tr>
<td>TSV (text/tab-separated-values)</td>
</tr>
<tr>
<td>RDF/Turtle (text/turtle)</td>
</tr>
<tr>
<td>JSON (application/json, application/*+json)</td>
</tr>
<tr>
<td>Programming languages (application/javascript, python files, etc.)</td>
</tr>
</tbody>
</table>

*Fragments are not formally defined through IETF, though there are well-known connections to existing fragments or practices

B. Complete Example

This section is non-normative.

Entirely Contrived Example Use Case: Juliet wants to associate a comment that she wrote in English within the annotation or an external mp3 of the same content in German by someone else, plus a tag, with a range of characters from a particular element in an XML representation of a document as it was at a certain point in time, and for it to be displayed in a particular way.
EXAMPLE 41: Complete Example
"@context": "http://www.w3.org/ns/anno.jsonld",
"id": "http://example.org/anno38",
"type": "Annotation",
"motivation": "commenting",
"creator": {
  "id": "http://example.org/user1",
  "type": "Person",
  "name": "A. Person",
  "nickname": "user1"
},
"created": "2015-10-13T13:00:00Z",
"generator": {
  "id": "http://example.org/client1",
  "type": "Software",
  "name": "Code v2.1",
  "homepage": "http://example.org/homepage1"
},
"generated": "2015-10-14T15:13:28Z",
"stylesheet": {
  "id": "http://example.org/stylesheet1",
  "type": "CssStylesheet"
},
"body": [
  {
    "type": "TextualBody",
    "purpose": "tagging",
    "value": "love"
  },
  {
    "type": "Choice",
    "items": [
      {
        "type": "TextualBody",
        "purpose": "describing",
        "value": "I really love this particular bit of text in this XML. No really.",
        "format": "text/plain",
        "language": "en",
        "creator": "http://example.org/user1"
      },
      {
        "type": "SpecificResource",
        "purpose": "describing",
        "source": {
          "id": "http://example.org/comment1",
          "type": "Audio",
          "format": "audio/mpeg",
          "language": "de",
          "creator": {
            "id": "http://example.org/user2",
            "type": "Person"
          }
        }
      }
    ]
  }
]
C. Index of JSON Keys

This section is non-normative.

<table>
<thead>
<tr>
<th>Key</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessibility</td>
<td>Body, Target</td>
</tr>
<tr>
<td>audience</td>
<td>Audience</td>
</tr>
<tr>
<td>body</td>
<td>Annotation</td>
</tr>
<tr>
<td>bodyValue</td>
<td>Annotation</td>
</tr>
<tr>
<td>cached</td>
<td>Time State</td>
</tr>
<tr>
<td>canonical</td>
<td>Annotation, Body, Target</td>
</tr>
<tr>
<td>conformsTo</td>
<td>Fragment Selector</td>
</tr>
<tr>
<td>created</td>
<td>Annotation, Body</td>
</tr>
<tr>
<td>creator</td>
<td>Annotation, Body</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>email</td>
<td>Agent</td>
</tr>
<tr>
<td>email_sha1</td>
<td>Agent</td>
</tr>
<tr>
<td>end</td>
<td>Text Position Selector, Data Position Selector</td>
</tr>
<tr>
<td>endSelector</td>
<td>Range Selector</td>
</tr>
<tr>
<td>exact</td>
<td>Text Quote Selector</td>
</tr>
<tr>
<td>first</td>
<td>Annotation Collection</td>
</tr>
<tr>
<td>format</td>
<td>Body, Target, SVG Selector</td>
</tr>
<tr>
<td>generated</td>
<td>Annotation</td>
</tr>
<tr>
<td>generator</td>
<td>Annotation</td>
</tr>
<tr>
<td>homepage</td>
<td>Agents</td>
</tr>
<tr>
<td>id</td>
<td>Note: Every object may have an id. Annotation, Body, Target, Segments of External Resources, Embedded Textual Body, Agent, Audience, Specific Resource</td>
</tr>
<tr>
<td>items</td>
<td>Choice, Annotation Page</td>
</tr>
<tr>
<td>label</td>
<td>Annotation Collection</td>
</tr>
<tr>
<td>language</td>
<td>Body, Target</td>
</tr>
<tr>
<td>last</td>
<td>Annotation Collection</td>
</tr>
<tr>
<td>modified</td>
<td>Annotation, Body</td>
</tr>
<tr>
<td>motivation</td>
<td>Annotation</td>
</tr>
<tr>
<td>name</td>
<td>Agent</td>
</tr>
<tr>
<td>nickname</td>
<td>Agent</td>
</tr>
<tr>
<td>next</td>
<td>Annotation Page</td>
</tr>
<tr>
<td>partOf</td>
<td>Annotation Page</td>
</tr>
<tr>
<td>prefix</td>
<td>Text Quote Selector</td>
</tr>
<tr>
<td>prev</td>
<td>Annotation Page</td>
</tr>
<tr>
<td>purpose</td>
<td>Textual Body, Specific Resource</td>
</tr>
<tr>
<td>renderedVia</td>
<td>Specific Resource</td>
</tr>
<tr>
<td>rights</td>
<td>Annotation, Body, Target</td>
</tr>
<tr>
<td>refinedBy</td>
<td>Selector, State</td>
</tr>
<tr>
<td>scope</td>
<td>Specific Resource</td>
</tr>
<tr>
<td>selector</td>
<td>Specific Resource</td>
</tr>
<tr>
<td>source</td>
<td>Specific Resource</td>
</tr>
<tr>
<td>sourceDate</td>
<td>Time State</td>
</tr>
</tbody>
</table>
D. Sets of Bodies and Targets

This section is non-normative.

While it is possible to annotate multiple targets, the meaning of that annotation is that each Body applies independently to each of the Targets. This might not be the intent of the annotator, such as when all of the targets are required for the annotation to be correctly understood. In order to allow annotators to capture these requirements, a resource similar to Choice could be used, such as a Composite (unordered) or List (ordered).

The technical implementation of this pattern is not difficult, as it is practically identical to Choice, however the implementation of a user interface that can manage a human user's interactions such that the client can recognize the distinctions has proven to be very challenging. As such, the pattern is noted in this appendix for future consideration.

Example Use Case: Karin comments on a set of web pages as, together, providing evidence towards her research hypothesis. Her client creates a Composite, as there is no inherent order to the set of web pages.

Example Use Case: Lana tags a list of pages within a book as being important. As the pages have an order in the book, her client creates a List to maintain that order.
Example Use Case: Melanie annotates a set of images to classify them as portraits. As the classification applies to each image independently, her client creates a Independents resource to group them.

Proposed Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Property</td>
<td>The IRI that identifies the set. The set resource <strong>may</strong> have exactly 1 IRI that identifies it.</td>
</tr>
<tr>
<td>type</td>
<td>Relationship</td>
<td>The type of the resource. The set <strong>must</strong> have exactly 1 type class, taken from the options below.</td>
</tr>
<tr>
<td>Composite</td>
<td>Class</td>
<td>A set of resources, all of which are required for the Annotation to be correctly interpreted.</td>
</tr>
<tr>
<td>List</td>
<td>Class</td>
<td>An ordered list of resources, all of which are required in order for the Annotation to be correctly interpreted.</td>
</tr>
<tr>
<td>Independents</td>
<td>Class</td>
<td>A set of resources, each of which is being annotated separately with the same interpretation as having multiple bodies or targets directly associated with the Annotation.</td>
</tr>
<tr>
<td>items</td>
<td>Relationship</td>
<td>The list of resources in the Composite, List, or Independents.</td>
</tr>
</tbody>
</table>

Examples

EXAMPLE 42: Composite

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno39",
    "type": "Annotation",
    "motivation": "commenting",
    "body": {
        "type": "TextualBody",
        "value": "These pages together provide evidence of the conspiracy"
    },
    "target": {
        "type": "Composite",
        "items": [
            "http://example.com/page1",
            "http://example.org/page6",
            "http://example.net/page4"
        ]
    }
}
```
E. Acknowledgments

This section is non-normative.

The Web Annotation Working Group gratefully acknowledges the contributions of the Open Annotation Community Group. The output of the Community Group was fundamental to the current data model. In particular the editors would like to thank

EXAMPLE 43: List

{
   "@context": "http://www.w3.org/ns/anno.jsonld",
   "id": "http://example.org/anno40",
   "type": "Annotation",
   "motivation": "tagging",
   "body": {
      "@type": "TextualBody",
      "value": "important"
   },
   "target": {
      "@type": "List",
      "items": [
         "http://example.com/book/page1",
         "http://example.com/book/page2",
         "http://example.com/book/page3",
         "http://example.com/book/page4"
      ]
   }
}

EXAMPLE 44: Independents

{
   "@context": "http://www.w3.org/ns/anno.jsonld",
   "id": "http://example.org/anno41",
   "type": "Annotation",
   "motivation": "classifying",
   "body": "http://example.org/vocab/art/portrait",
   "target": {
      "@type": "Independents",
      "items": [
         "http://example.com/image1",
         "http://example.net/image2",
         "http://example.com/image4",
         "http://example.org/image9"
      ]
   }
}
Herbert Van de Sompel of Los Alamos National Laboratory for his editorial contributions throughout the Community Group process.

The following people have been instrumental in providing thoughts, feedback, reviews, content, criticism and input in the creation of this specification:


F. Candidate Recommendation Exit Criteria

This section is non-normative.

For this specification to be advanced to Proposed Recommendation, there must be at least two independent implementations of each feature described below. Each feature may be implemented by a different set of products, and there is no requirement that any single product implement every feature.

Features

For the purposes of evaluating exit criteria, the following are considered as features:

- The Annotation class and required properties.
- The Agent class and required properties, as related to an Annotation.
- The Agent class and required properties, as related to a resource used as the Body of an Annotation.
- Embedded TextualBody class and required properties.
- External web resources, used as the Body of an Annotation.
- A Choice of resources, used as the Body of an Annotation.
- The SpecificResource class and required properties, used as the Body of an Annotation.
- External web resources, used as the Target of an Annotation.
- The SpecificResource class and required properties, used as the Target of an Annotation.
- The AnnotationCollection class and required properties.
- The AnnotationPage class and required properties.

Software that does not alter its behavior in the presence or lack of a given feature is not deemed to implement that feature for the purposes of exiting the Candidate recommendation phase.

G. Changes from Previous Versions

This section is non-normative.
G.1 Changes from the Proposed Recommendation of 2017-01-17

No significant changes.

G.2 Changes from the Candidate Recommendation of 2016-11-22

- Removed unnecessary appendix
- Updated SVG reference to 1.1

G.3 Changes from the Candidate Recommendation of 2016-09-06

- Move Composite, List and Independents classes to an informative appendix. (These classes were marked as "at Risk" in earlier versions.)
- Remove the use of Choice with Targets, and associating Agents with Targets.
- Clarify the possibility of embedding Collection information in a Page using partOf.
- Improve description of multiple rights statements about resources.
- Improve description of textDirection and note that value definitions are explicitly from HTML5, as well as the property.
- Link to the media type in protocol from the introduction.
- Clarified that processingLanguage should also have a BCP47 formatted value.
- Clarified that TextualBody can be the source of a Specific Resource, and the target of an Annotation.
- Add informative principle that features of externally referenced resources described in an Annotation are not intended to be authoritative, and the remote resource is the source of truth about itself.
- Clarified the scope of the purpose property with respect to TextualBody resources.

G.4 Changes from the Candidate Recommendation of 2016-07-05

- Added CR Exit Criteria

G.5 Changes from the Working Draft of 2016-03-31

Significant technical changes in this specification from the Working Draft Published of 2016-03-31 are:

- Addition of conformance section and media table
- Remove Content class as unnecessary, and reuse value in place of the previous distinct text property.
- Rename bodyText to bodyValue following from the removal of the text property.
- Simplification of SvgSelector and CssStylesheet, following from the removal of Content.
- Add accessibility section.
- Clarify normalization requirements for text based selectors, and add textDirection and processingLanguage properties.
- Add Independents class to mirror the use of of multiple bodies or targets. Independents, and the re-introduced Composite and List, marked at risk.
- Revert to the use of xsd:dateTime for dates.
- Clarify the semantics of multiple selectors, states.
- Use less ambiguous nickname instead of account for foaf:nick.
- Remove role requirement for interpretation of bodyValue.

G.6 Changes from the Open Annotation Draft

Significant technical changes in this specification from the Open Annotation Community Group's draft are:

- Use intuitive, memorable and developer-friendly names for the JSON-LD keys. (text)
- Replace the ContentAsText construction which was not taken through the standardization process. (text)
- Allow a string literal as the body via bodyText. (text)
- Allow an ordered list of options for Choice (text).
- Add additional core lifecycle metadata for resources, including rights information and intended audience. (text)
- Align identity equivalence with other standards. (text)
- Allow association of Motivation as roles/purposes on a per body or target basis, including alignment of Tags and Semantic Tags. (text)
- Introduce CSS Selector, XPath Selector and Range Selector from implementation experience.
- Use structure rather than resources for multiple specifiers. (text)
- Add the capability to describe rendering software. (text)
- Add Collections of Annotations as a defined pattern.
- Separate the ontology [annotation-vocab] from the model and JSON serialization.
- Deprecate embedded graphs as an explicit part of the model, instead just include or reference a serialized graph.

H. References

H.1 Normative references

[CSS3-selectors]
Selectors Level 3. Tantek Çelik; Elika Etemad; Daniel Glazman; Ian Hickson; Peter Linss; John Williams et al. W3C. 29 September 2011. W3C Recommendation. URL: https://www.w3.org/TR/css3-selectors/

[DOM-Level-3-XPath]

[JSON-LD]
JSON-LD 1.0. Manu Sporny; Gregg Kellogg; Markus Lanthaler. W3C. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/json-ld/
[RFC2119]

[SVG11]
Scalable Vector Graphics (SVG) 1.1 (Second Edition). Erik Dahlström; Patrick Dengler; Anthony Grasso; Chris Lilley; Cameron McCormack; Doug Schepers; Jonathan Watt; Jon Ferraiole; Jun Fujisawa; Dean Jackson et al. W3C. 16 August 2011. W3C Recommendation. URL: https://www.w3.org/TR/SVG11/

[annotation-protocol]

[annotation-vocab]
Web Annotation Vocabulary. Robert Sanderson; Paolo Ciccarese; Benjamin Young. W3C. W3C Recommendation. URL: http://www.w3.org/TR/annotation-vocab/

[bcp47]

[cfi]
EPUB Canonical Fragment Identifiers. Peter Sorotokin; Garth Conboy; Brady Duga; John Rivlin; Don Beaver; Kevin Ballard; Alastair Fettes; Daniel Weck. IDPF. Recommended Specification. URL: http://www.idpf.org/epub/linking/cfi/epub-cfi-20140628.html

[charmod]
Character Model for the World Wide Web 1.0: Fundamentals. Martin Dürst; François Yergeau; Richard Ishida; Misha Wolf; Tex Texin et al. W3C. 15 February 2005. W3C Recommendation. URL: https://www.w3.org/TR/charmod/

[fragid-best-practices]

[media-frags]
Media Fragments URI 1.0 (basic). Raphaël Troncy; Erik Mannens; Silvia Pfeiffer; Davy Van Deursen. W3C. 25 September 2012. W3C Recommendation. URL: https://www.w3.org/TR/media-frags/

[rfc3023]

[rfc3236]

[rfc3778]

[rfc3870]

[rfc3987]

[rfc5147]
H.2 Informative references

[html5]  
**HTML5.** Ian Hickson; Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Theresa O’Connor; Silvia Pfeiffer. W3C. 28 October 2014. W3C Recommendation. URL: [https://www.w3.org/TR/html5/](https://www.w3.org/TR/html5/)

[iana-media-types]  
**Media Types.** IANA (Internet Assigned Numbers Authority). URL: [http://www.iana.org/assignments/media-types/](http://www.iana.org/assignments/media-types/)

[w3c-language-tags]  
**Language Tags in HTML and XML.** W3C. URL: [https://www.w3.org/International/articles/language-tags/](https://www.w3.org/International/articles/language-tags/)
Abstract

The Web Annotation Vocabulary specifies the set of RDF classes, predicates and named entities that are used by the Web Annotation Data Model [annotation-model]. It also lists recommended terms from
other ontologies that are used in the model, and provides the JSON-LD Context and profile definitions needed to use the Web Annotation JSON serialization in a Linked Data context.

**Status of This Document**

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

By publishing this Recommendation, W3C expects that the functionality specified in this Recommendation will not be affected by changes to the Activity Streams 2.0 [activitystreams-core] and Activity Vocabulary [activitystreams-vocabulary] as those specifications proceed to Recommendation.

This document was published by the Web Annotation Working Group as a Recommendation. If you wish to make comments regarding this document, please send them to public-annotation@w3.org (subscribe, archives). All comments are welcome.

Please see the Working Group's implementation report.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 September 2015 W3C Process Document.

**Table of Contents**

1. **Introduction**
   1.1 Namespaces
   1.2 Diagrams and Examples
   1.3 Conformance
1.4 Terminology

2. Web Annotation Ontology
2.1 Classes
2.1.1 Annotation
2.1.2 Choice
2.1.3 CssSelector
2.1.4 CssStyle
2.1.5 DataPositionSelector
2.1.6 Direction
2.1.7 FragmentSelector
2.1.8 HttpRequestState
2.1.9 Motivation
2.1.10 RangeSelector
2.1.11 ResourceSelection
2.1.12 Selector
2.1.13 SpecificResource
2.1.14 State
2.1.15 Style
2.1.16 SvgSelector
2.1.17 TextPositionSelector
2.1.18 TextQuoteSelector
2.1.19 TextualBody
2.1.20 TimeState
2.1.21 XPathSelector
2.2 Properties
2.2.1 annotationService
2.2.2 bodyValue
2.2.3 cachedSource
2.2.4 canonical
2.2.5 end
2.2.6 exact
2.2.7 hasBody
2.2.8 hasEndSelector
2.2.9 hasPurpose
2.2.10 hasScope
2.2.11 hasSelector
2.2.12 hasSource
2.2.13 hasStartSelector
2.2.14 hasState
2.2.15 hasTarget
2.2.16 motivatedBy
2.2.17 prefix
2.2.18 processingLanguage
2.2.19 refinedBy
2.2.20 renderedVia
2.2.21 sourceDate
2.2.22 sourceDateEnd
2.2.23 sourceDateStart
2.2.24 start
2.2.25 styleClass
2.2.26 styledBy
2.2.27 suffix
2.2.28 textDirection
2.2.29 via

2.3 Named Individuals
2.3.1 assessing
2.3.2 bookmarking
2.3.3 classifying
2.3.4 commenting
2.3.5 describing
2.3.6 editing
2.3.7 highlighting
2.3.8 identifying
2.3.9 linking
2.3.10 moderating
2.3.11 questioning
2.3.12 replying
2.3.13 tagging
2.3.14 autoDirection
2.3.15 ltrDirection
2.3.16 rtlDirection
2.3.17 PreferContainedDescriptions
2.3.18 PreferContainedIRIs

3. Recommended Ontologies
3.1 Classes
3.1.1 as:Application
as:OrderedCollection
as:OrderedCollectionPage
dctypes:Dataset
dctypes:MovingImage
dctypes:StillImage
dctypes:Sound
dctypes:Text
foaf:Organization
foaf:Person
schema:Audience

Properties
as:first
as:generator
as:items
as:last
as:next
as:partOf
as:prev
as:startIndex
as:totalItems
dc:format
dc:language
dcterms:conformsTo
dcterms:created
dcterms:creator
dcterms:issued
dcterms:modified
dcterms:rights
foaf:homepage
foaf:mbox
foaf:mbox_sha1sum
foaf:name
foaf:nick
rdf:type
rdf:value
rdfs:label
schema:accessibilityFeature
schema:audience
1. Introduction

This section is non-normative.

The specification is divided into two major sections: the terms defined in the Web Annotation ontology, and terms from other ontologies used in the model.

Each class lists the recommendations from the model for the required, recommended and optional object and data properties for instances of the class. Instances may, of course, be the subject of any other
triples that implementers find useful, however there is no expectation of interoperability in these cases.

1.1 Namespaces

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oa</td>
<td><a href="http://www.w3.org/ns/oa#">http://www.w3.org/ns/oa#</a></td>
<td>The Web Annotation Data Model</td>
</tr>
<tr>
<td>as</td>
<td><a href="http://www.w3.org/ns/activitystreams#">http://www.w3.org/ns/activitystreams#</a></td>
<td>[activitystreams-vocabulary]</td>
</tr>
<tr>
<td>dcterms</td>
<td><a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a></td>
<td>[DC-TERMS]</td>
</tr>
<tr>
<td>dctypes</td>
<td><a href="http://purl.org/dc/dcmitype/">http://purl.org/dc/dcmitype/</a></td>
<td>[DC-TERMS]</td>
</tr>
<tr>
<td>foaf</td>
<td><a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a></td>
<td>[FOAF]</td>
</tr>
<tr>
<td>rdf</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
<td>[rdf-schema]</td>
</tr>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>[rdf-schema]</td>
</tr>
<tr>
<td>schema</td>
<td><a href="http://schema.org/">http://schema.org/</a></td>
<td>schema.org</td>
</tr>
<tr>
<td>skos</td>
<td><a href="http://www.w3.org/2004/02/skos/core#">http://www.w3.org/2004/02/skos/core#</a></td>
<td>[skos-reference]</td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
<td>[xmlschema-2]</td>
</tr>
</tbody>
</table>

1.2 Diagrams and Examples

The examples throughout the document are serialized as [Turtle] with the prefixes taken from the namespace declarations given in Appendix A. The examples are informative only.

The diagrams use the following style

- Instances are depicted as colored ellipses.
- Instances without an IRI are depicted as colored ellipses with double lines.
- Classes are depicted as white rectangles.
- Literals are depicted as white lozenges.
- Relationships and properties are depicted as straight, black lines.
- Class instantiation is depicted as a straight black line with white arrow head.
Example instance identifiers are lowercase and end in a number. For example, `anno1` is a specific instance of an Annotation, whereas `Annotation` is a class.

Example literals follow the requirements for the model and, thus, must not be interpreted as the only possible value.

Lists are depicted as vertical braces with a gray background and '...' in the middle (regardless of if there are actually other items in the list or not).

Conceptual resource boundaries not explicit in the model, but considered important for understanding, are depicted as gray dashed boxes around the components. They are used to convey spatial parts of the diagrams and may be safely ignored.

### 1.3 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words `may`, `must`, `must not`, `optional`, `recommended`, `required`, and `should` are to be interpreted as described in [RFC2119].

### 1.4 Terminology

**IRI**

An **IRI**, or Internationalized Resource Identifier, is an extension to the URI specification to allow characters from Unicode, whereas URIs must be made up of a subset of ASCII characters. There is a mapping algorithm for translating between IRIs and the equivalent encoded URI form. IRIs are defined by [rfc3987].

**Resource**

An item of interest that **may** be identified by an **IRI**.

---

**NOTE**

RDF [rdf-concepts] is fully internationalized and permits the use of IRIs for identifying resources. This specification uses the term IRI to make it clear that this is the case.

### 2. Web Annotation Ontology

The namespace used for the Web Annotation Ontology is: [http://www.w3.org/ns/oa#](http://www.w3.org/ns/oa#)
2.1 Classes

2.1.1 Annotation

The class for Web Annotations.

**IRI:** http://www.w3.org/ns/oa#Annotation

**Required Predicates:** `oa:hasTarget`, `rdf:type`

**Recommended Predicates:** `oa:hasBody`, `oa:motivatedBy`, `dcterms:creator`, `dcterms:created`

**Other Predicates:** `oa:bodyValue`, `oa:styledBy`, `dcterms:issued`, `as:generator`
2.1.2 Choice

A subClass of `as:OrderedCollection` that conveys to a consuming application that it should select one of the resources in the `as:items` list to use, rather than all of them. This is typically used to provide a choice of resources to render to the user, based on further supplied properties. If the consuming application cannot determine the user's preference, then it should use the first in the list.

**IRI:** http://www.w3.org/ns/oa#Choice

**Sub Class Of:** `as:OrderedCollection`

**Required Predicates:** `as:items`, `rdf:type`
2.1.3 CssSelector

A CssSelector describes a Segment of interest in a representation that conforms to the Document Object Model through the use of the CSS selector specification.

IRI: http://www.w3.org/ns/oa#CssSelector

Sub Class Of: oa:Selector

Required Predicates: rdf:value, rdf:type
EXAMPLE 3: oa:CdSelector

```xml
<http://example.org/anno3> a oa:Annotation ;
  oa:hasBody <http://example.org/note1> ;
  oa:hasTarget [
    oa:hasSource <http://example.org/page1> ;
    oa:hasSelector [
      a oa:CdSelector ;
```

### 2.1.4 CssStyle

A resource which describes styles for resources participating in the Annotation using CSS.

- **IRI:** http://www.w3.org/ns/oa#CssStyle
- **Sub Class Of:** [oa:Style](http://www.w3.org/ns/oa#Style)
2.1.5 DataPositionSelector

DataPositionSelector describes a range of data by recording the start and end positions of the selection in the stream. Position 0 would be immediately before the first byte, position 1 would be immediately before the second byte, and so on. The start byte is thus included in the list, but the end byte is not.

**IRI:** http://www.w3.org/ns/oa#DataPositionSelector

**EXAMPLE 4**

```xml
<http://example.org/anno4> a oa:Annotation ;
    oa:hasBody <http://example.org/comment1> ;
    oa:styledBy <http://example.org/style1> ;
    oa:hasTarget [ 
        oa:hasSource <http://example.org/document1> ;
        oa:styleClass "red" ] .

<http://example.org/style1> a oa:CssStyle .
```
2.1.6 Direction

A class to encapsulate the different text directions that a textual resource might take. It is not used directly in the Annotation Model, only its three instances.
2.1.7 FragmentSelector

The FragmentSelector class is used to record the segment of a representation using the IRI fragment specification defined by the representation's media type.
EXAMPLE 7: oa:FragmentSelector

<http://example.org/anno7> a oa:Annotation ;
  oa:hasBody <http://example.org/image1> ;
  oa:hasTarget [
    oa:hasSource <http://example.org/video1> ;
    oa:hasSelector [
      a oa:FragmentSelector ;
      dcterms:conformsTo <http://www.w3.org/TR/media-frags/> ;
      rdf:value "t=30,60" ] ] .
The HttpRequestState class is used to record the HTTP request headers that a client should use to request the correct representation from the resource.

**IRI:** http://www.w3.org/ns/oa#HttpRequestState

**Sub Class Of:** oa:State

**Required Predicates:** rdf:value, rdf:type

**EXAMPLE 8:** oa:HttpRequestState

```xml
<http://example.org/anno8> a oa:Annotation ;
    oa:hasBody <http://example.org/description1> ;
    oa:hasTarget [ oa:hasSource <http://example.org/target1> ;
                  oa:hasState [ a oa:HttpRequestState ;
```
2.1.9 Motivation

The Motivation class is used to record the user's intent or motivation for the creation of the Annotation, or the inclusion of the body or target, that it is associated with.

**IRI:** http://www.w3.org/ns/oa#Motivation

**Sub Class Of:** skos:Concept

**Range Of:** oa:motivatedBy, oa:hasPurpose

---

EXAMPLE 9: oa:Motivation

```xml
<http://example.org/anno9> a oa:Annotation ;
  oa:hasBody <http://example.org/description1> ;
  oa:hasTarget <http://example.com/resource1> ;
  oa:motivatedBy oa:describing .
```

---

2.1.10 RangeSelector

A Range Selector can be used to identify the beginning and the end of the selection by using other Selectors. The selection consists of everything from the beginning of the starting selector through to the beginning of the ending selector, but not including it.
IRI: http://www.w3.org/ns/oa#RangeSelector

Sub Class Of: oa:Selector

Required Predicates: oa:hasStartSelector, oa:hasEndSelector, rdf:type
2.1.11 ResourceSelection

Instances of the ResourceSelection class identify part (described by an oa:Selector) of another resource (referenced with oa:hasSource), possibly from a particular representation of a resource (described by an oa:State). Please note that ResourceSelection is not used directly in the Web Annotation model, but is provided as a separate class for further application profiles to use, separate from oa:SpecificResource which has many Annotation specific features.

IRI: http://www.w3.org/ns/oa#ResourceSelection

**Required Predicates:** [oa:hasSource]

**Recommended Predicates:** [oa:hasSelector, oa:hasState]

NOTE: oa:Selector

No example is given. The class should not be used in the Web Annotation model directly.

2.1.12 Selector

**EXAMPLE 10: oa:RangeSelector**

```html
<http://example.org/anno10> a oa:Annotation ;
   oa:hasBody <http://example.org/note1> ;
   oa:hasTarget [
       oa:hasSource <http://example.org/page1> ;
       oa:hasSelector [a oa:RangeSelector ;
                        oa:hasStartSelector [a oa:XPathSelector ;
                                        rdf:value "/table[1]/tr[1]/td[2]" ] ;
                        oa:hasEndSelector [a oa:XPathSelector ;
```
A resource which describes the segment of interest in a representation of a Source resource, indicated with oa:hasSelector from the Specific Resource. This class is not used directly in the Annotation model, only its subclasses.

**IRI:** http://www.w3.org/ns/oa#Selector

**Super Class Of:** oa:FragmentSelector, oa:TextQuoteSelector, oa:TextPositionSelector, oa:DataPositionSelector, oa:SvgSelector

**Range Of:** oa:hasSelector

---

**NOTE:** oa:Selector

No example is given. The class should only be used to derive subclasses.

---

**2.1.13 SpecificResource**

Instances of the SpecificResource class identify part of another resource (referenced with oa:hasSource), a particular representation of a resource, a resource with styling hints for renders, or any combination of these, as used within an Annotation.

**IRI:** http://www.w3.org/ns/oa#SpecificResource

**Sub Class Of:** oa:ResourceSelection

**Required Predicates:** oa:hasSource

**Recommended Predicates:** oa:hasSelector, oa:hasState, oa:hasPurpose

**Other Predicates:** oa:hasScope, oa:styleClass
2.1.14 State

A State describes the intended state of a resource as applied to the particular Annotation, and thus provides the information needed to retrieve the correct representation of that resource.

**IRI:** http://www.w3.org/ns/oa#State

**Super Class Of:** `oa:HttpRequestState`, `oa:TimeState`

**Range Of:** `oa:hasState`
2.1.15 Style

A Style describes the intended styling of a resource as applied to the particular Annotation, and thus provides the information to ensure that rendering is consistent across implementations.

IRI: http://www.w3.org/ns/oa#Style

Super Class Of: oa.CssStyle

Range Of: oa:styledBy

---

NOTE: oa:Style

No example is given. The class should only be used in further ontologies to derive subclasses.

2.1.16 SvgSelector

An SvgSelector defines an area through the use of the Scalable Vector Graphics [SVG] standard. This allows the user to select a non-rectangular area of the content, such as a circle or polygon by describing the region using SVG. The SVG may be either embedded within the Annotation or referenced as an External Resource.

IRI: http://www.w3.org/ns/oa#SvgSelector

Sub Class Of: oa:Selector

Required Predicates: rdf:type

Other Predicates: rdf:value
2.1.17 TextPositionSelector

The TextPositionSelector describes a range of text by recording the start and end positions of the selection in the stream. Position 0 would be immediately before the first character, position 1 would be immediately before the second character, and so on.

IRI: http://www.w3.org/ns/oa#TextPositionSelector

Sub Class Of: oa:Selector

Required Predicates: oa:start, oa:end, rdf:type

EXAMPLE 12: oa:SvgSelector

<http://example.org/anno12> a oa:Annotation ;
  oa:hasBody <http://example.org/road1> ;
  oa:hasTarget [ oas:hasSource <http://example.org/map1> ;
  oas:hasSelector [ a oas:SvgSelector ;

2.1.17 TextPositionSelector

The TextPositionSelector describes a range of text by recording the start and end positions of the selection in the stream. Position 0 would be immediately before the first character, position 1 would be immediately before the second character, and so on.

IRI: http://www.w3.org/ns/oa#TextPositionSelector

Sub Class Of: oa:Selector

Required Predicates: oa:start, oa:end, rdf:type
The TextQuoteSelector describes a range of text by copying it, and including some of the text immediately before (a prefix) and after (a suffix) it to distinguish between multiple copies of the same sequence of characters.

**IRI:** http://www.w3.org/ns/oa#TextQuoteSelector
EXAMPLE 14: oa:TextQuoteSelector

```
<http://example.org/anno14> a oa:Annotation ;
   oa:hasBody <http://example.org/comment1> ;
   oa:hasTarget [
      oa:hasSource <http://example.org/page1> ;
      oa:hasSelector [
         a oa:TextQuoteSelector ;
         oa:exact "annotation" ;
         oa:prefix "this is an " ;
         oa:suffix " that has some" ] ] .
```

2.1.19 TextualBody
EXAMPLE 15: oa:TextualBody

```
<http://example.org/anno15> a oa:Annotation ;
    oa:hasTarget <http://example.org/photo1> ;
    oa:hasBody [ a oa:TextualBody;
        rdf:value "<p>Comment text</p>" ;
        dc:language "en" ;
        dc:format "text/html" ].
```

2.1.20 TimeState
A TimeState records the time at which the resource's state is appropriate for the Annotation, typically the time that the Annotation was created and/or a link to a persistent copy of the current version.

**IRI:** http://www.w3.org/ns/oa#TimeState

**Sub Class Of:** [oa:State](http://www.w3.org/ns/oa#State)

**Required Predicates:** [rdf:type](http://www.w3.org/1999/02/22-rdf-syntax-ns#type)

**Recommended Predicates:** [oa:sourceDate](http://www.w3.org/ns/oa#sourceDate), [oa:cachedSource](http://www.w3.org/ns/oa#cachedSource)
2.1.21 XPathSelector

An XPathSelector is used to select elements and content within a resource that supports the Document Object Model via a specified XPath value.

---

**IRI:** http://www.w3.org/ns/oa#XPathSelector

**Sub Class Of:** [oa:Selector](#)

**Required Predicates:** [rdf:value](http://www.w3.org/1999/02/22-rdf-syntax-ns#value), [rdf:type](http://www.w3.org/1999/02/22-rdf-syntax-ns#type)

---

**EXAMPLE 16: oa:TimeState**

```xml
<http://example.org/anno16> a oa:Annotation ;
    oa:hasBody <http://example.org/note1> ;
    oa:hasTarget [
        oa:hasSource <http://example.org/page1> ;
        oa:hasState [a oa:TimeState ;
            oa:cachedSource <http://example.org/copy1> ;
            oa:sourceDate "2015-07-20T13:30:00Z" ] ] .
```
EXAMPLE 17: oa:XPathSelector

```xml
<http://example.org/anno17> a oa:Annotation ;
  oa:hasBody  <http://example.org/note1> ;
  oa:hasTarget [
    oa:hasSource  <http://example.org/page1> ;
    oa:hasSelector [
      a oa:XPathSelector ;
```

2.2 Properties

annotationService | bodyValue | cachedSource | canonical | end | exact | hasBody | hasPurpose | hasScope | hasSelector | hasSource | hasState | hasTarget | motivatedBy | prefix | processingLanguage | refinedBy | sourceDate | sourceDateEnd | sourceDateStart | start | styleClass | styledBy | suffix | textDirection | via
### 2.2.1 annotationService

The object of the relationship is the end point of a service that conforms to the [annotation-protocol], and it may be associated with any resource. The expectation of asserting the relationship is that the object is the preferred service for maintaining annotations about the subject resource, according to the publisher of the relationship.

This relationship is intended to be used both within Linked Data descriptions and as the rel type of a Link, via HTTP Link Headers [rfc5988] for binary resources and in HTML <link> elements. For more information about these, please see the Annotation Protocol specification [annotation-protocol].

**IRI:** http://www.w3.org/ns/oa#annotationService

**EXAMPLE 18: oa:annotationService**

```xml
<http://example.org/diagram.jpg> a dctypes:Image ;
   oa:annotationService <http://example.org/services/annotations/> .
```

### 2.2.2 bodyValue

The object of the predicate is a plain text string to be used as the content of the body of the Annotation. The value must be an xsd:string and that data type must not be expressed in the serialization. Note that language must not be associated with the value either as a language tag, as that is only available for rdf:langString.

**IRI:** http://www.w3.org/ns/oa#bodyValue

**Domain:** oa:Annotation

**Range:** xsd:string
### 2.2.3 cachedSource

A object of the relationship is a copy of the Source resource's representation, appropriate for the Annotation.

**IRI:** http://www.w3.org/ns/oa#cachedSource

**Domain:** oa:TimeState
2.2.4 canonical

A object of the relationship is the canonical IRI that can always be used to deduplicate the Annotation, regardless of the current IRI used to access the representation.

**IRI:** http://www.w3.org/ns/oa#canonical
2.2.5 end

The end property is used to convey the 0-based index of the end position of a range of content.

IRI: http://www.w3.org/ns/oa#end

Range: xsd:nonNegativeInteger
2.2.6 exact

The object of the predicate is a copy of the text which is being selected, after normalization.

**IRI:** http://www.w3.org/ns/oa#exact

**Range:** xsd:string
2.2.7 hasBody

The object of the relationship is a resource that is a body of the Annotation.

IRI: http://www.w3.org/ns/oa#hasBody
EXAMPLE 24: oa:hasBody

```
<http://example.org/anno23> a oa:Annotation ;
  oa:hasBody <http://example.org/post1> ;
  oa:hasTarget <http://example.com/page1> .
```

2.2.8 hasEndSelector

The relationship between a RangeSelector and the Selector that describes the end position of the range.

IRI: http://www.w3.org/ns/oa#hasEndSelector

Domain: oa:RangeSelector

Range: oa:Selector
EXAMPLE 25: oa:hasEndSelector

```
<http://example.org/anno24> a oa:Annotation ;
    oa:hasBody <http://example.org/note1> ;
    oa:hasTarget [
        oa:hasSource <http://example.org/page1> ;
        oa:hasSelector [
            a oa:RangeSelector ;
            oa:hasStartSelector [
                a oa:XPathSelector ;
                rdf:value "//table[1]/tr[1]/td[2]" ] ;
            oa:hasEndSelector [
                a oa:XPathSelector ;
```
The purpose served by the resource in the Annotation.

**IRI:** http://www.w3.org/ns/oa#hasPurpose

**Range:** oa:Motivation

**EXAMPLE 26:** oa:hasPurpose

```xml
<http://example.org/anno25> a oa:Annotation;
    oa:motivatedBy oa:bookmarking;
    oa:hasBody [ a oa:TextualBody;
                   rdf:value "readme";
                   oa:hasPurpose oa:tagging ];
    oa:hasTarget <http://example.org/page1> .
```

### 2.2.10 hasScope

The scope or context in which the resource is used within the Annotation.
2.2.11 hasSelector

The object of the relationship is a Selector that describes the segment or region of interest within the source resource. Please note that the domain (oa:ResourceSelection) is not used directly in the Web Annotation model.

IRI: http://www.w3.org/ns/oa#hasSelector
Domain: `oa:ResourceSelection`

Range: `oa:Selector`

EXAMPLE 28: `oa:hasSelector`

```
<http://example.org/anno27> a oa:Annotation ;
  oa:hasBody []
    oa:hasSource <http://example.org/page1> ;
    oa:hasSelector <http://example.org/paraselector1> ] ;
  oa:hasTarget []
    oa:hasSource <http://example.com/dataset1> ;
    oa:hasSelector <http://example.org/dataselector1> ] .
```

### 2.2.12 `hasSource`

The resource that the ResourceSelection, or its subclass SpecificResource, is refined from, or more specific than. Please note that the domain (`oa:ResourceSelection`) is not used directly in the Web Annotation model.

IRI: http://www.w3.org/ns/oa#hasSource
EXAMPLE 29: oa:hasSource

```
<http://example.org/anno28> a oa:Annotation ;
    oa:hasBody  <http://example.org/comment1> ;
    oa:hasTarget <http://example.org/region1> .

<http://example.org/region1> a oa:SpecificResource ;
    oa:hasSource <http://example.org/image1> .
```

### 2.2.13 hasStartSelector

The relationship between a RangeSelector and the Selector that describes the start position of the range.

**IRI:** http://www.w3.org/ns/oa#hasStartSelector

**Domain:** oa:RangeSelector

**Range:** oa:Selector
EXAMPLE 30: oa:hasStartSelector

```xml
<http://example.org/anno29> a oa:Annotation ;
  oa:hasBody <http://example.org/note1> ;
  oa:hasTarget [
    oa:hasSource <http://example.org/page1> ;
    oa:hasSelector [
      a oa:RangeSelector ;
      oa:hasStartSelector [
        a oa:XPathSelector ;
        rdf:value "//table[1]/tr[1]/td[2]" ] ;
      oa:hasEndSelector [
        a oa:XPathSelector ;
```
The relationship between the ResourceSelection, or its subclass SpecificResource, and a State resource. Please note that the domain (oa:ResourceSelection) is not used directly in the Web Annotation model.

**IRI:** http://www.w3.org/ns/oa#hasState

**Domain:** oa:ResourceSelection

**Range:** oa:State

**EXAMPLE 31:**

```
<http://example.org/anno30> a oa:Annotation ;
  oa:hasBody <http://example.org/note1> ;
  oa:hasTarget [ 
    oa:hasState <http://example.org/state1> ;
    oa:hasSource <http://example.org/page1> ] .
```

**2.2.15 hasTarget**

The relationship between an Annotation and its Target.
2.2.16 motivatedBy

The relationship between an Annotation and a Motivation that describes the reason for the Annotation's creation.

IRI: http://www.w3.org/ns/oa#motivatedBy

Domain: oa:Annotation

Range: oa:Motivation
The object of the property is a snippet of content that occurs immediately before the content which is being selected by the Selector.

**IRI:** http://www.w3.org/ns/oa#prefix

**Range:** xsd:string
The object of the property is the language that should be used for textual processing algorithms when dealing with the content of the resource, including hyphenation, line breaking, which font to use for rendering and so forth. The value must follow the recommendations of [BCP47].
EXAMPLE 35: oa:processingLanguage

```xml
<http://example.org/anno34> a oa:Annotation ;
  oa:hasBody [ a oa:TextualBody ;
    rdf:value "<p>הבינאום, פעילות ו3C</p>" ;
    dc:format "text/html" ;
    dc:language "en", "he" ;
    oa:processingLanguage "he" ] ;
  oa:hasTarget <http://example.org/page1> .
```

2.2.19 refinedBy
The relationship between a Selector and another Selector or a State and a Selector or State that should be applied to the results of the first to refine the processing of the source resource.

**IRI:** http://www.w3.org/ns/oa#refinedBy
EXAMPLE 36: oa:refinedBy

```xml
<http://example.org/anno35> a oa:Annotation ;
   oa:hasBody <http://example.org/comment1> ;
   oa:hasTarget [
      a oa:SpecificResource ;
      oa:hasSource <http://example.org/page1> ;
      oa:hasSelector [
         a oa:FragmentSelector ;
         rdf:value "para5" ;
         oa:refinedBy [
            a oa:TextQuoteSelector ;
            oa:exact "selected text" ;
            oa:prefix "text before the " ;
```

2.2.20 renderedVia

A system that was used by the application that created the Annotation to render the resource.

---------------------------

IRI: http://www.w3.org/ns/oa#renderedVia

Domain: oa:SpecificResource

---------------------------
2.2.21 sourceDate

The timestamp at which the Source resource should be interpreted as being applicable to the Annotation.

**IRI:** http://www.w3.org/ns/oa#sourceDate
The end timestamp of the interval over which the Source resource should be interpreted as being applicable to the Annotation.
EXAMPLE 39: oa:sourceDateEnd

```xml
<http://example.org/anno38> a oa:Annotation ;
    oa:hasBody <http://example.org/note1> ;
    oa:hasTarget [
        oa:hasSource <http://example.org/page1> ;
        oa:hasState [
            a oa:TimeState ;
            oa:sourceDateStart "2015-07-20T13:30:00Z" ;
            oa:sourceDateEnd "2015-07-21T19:45:00Z" ] ] .
```
2.2.23 sourceDateStart

The start timestamp of the interval over which the Source resource should be interpreted as being applicable to the Annotation.

IRI: http://www.w3.org/ns/oa#sourceDate

Domain: oa:TimeState

Range: xsd:dateTime
2.2.24 start

The start position in a 0-based index at which a range of content is selected from the data in the source resource.

**IRI:** http://www.w3.org/ns/oa#start

**Range:** xsd:nonNegativeInteger
2.2.25 styleClass

The name of the class used in the CSS description referenced from the Annotation that should be applied to the Specific Resource.

IRI: http://www.w3.org/ns/oa#styleClass
Domain: `oa:SpecificResource`  
Range: `xsd:string`  

**EXAMPLE 42: `oa:styleClass`**

```
<http://example.org/anno41> a `oa:Annotation` ;
  `oa:hasBody` <http://example.org/comment1> ;
  `oa:styledBy` <http://example.org/style1> ;
  `oa:hasTarget` [  
    `oa:hasSource` <http://example.org/document1> ;
    `oa:styleClass` "red" ] .
```

```
<http://example.org/style1> a `oa:CssStyle` .
```

---

### 2.2.26 `styledBy`

A reference to a Stylesheet that should be used to apply styles to the Annotation rendering.
EXAMPLE 43: `oa:styledBy`

```xml
<http://example.org/anno42> a `oa:Annotation`
    `oa:hasBody` <http://example.org/body1>
    `oa:styledBy` [
        a `oa:CssStyle`
        `rdf:value` "\red { color: red }"
    ]
    `oa:hasTarget` [
        `oa:hasSource` <http://example.org/target1>
        `oa:styleClass` "red"
    ]
```
EXAMPLE 44: oa:suffix

```
<http://example.org/anno43> a oa:Annotation ;
  oa:hasBody <http://example.org/comment1> ;
  oa:hasTarget [
    oa:hasSource <http://example.org/page1> ;
    oa:hasSelector [
      a oa:TextQuoteSelector ;
      oa:exact "annotation" ;
      oa:prefix "this is an " ;
      oa:suffix " that has some" ] ] .
```
2.2.28 textDirection

The direction of the text of the subject resource. There must only be one text direction associated with any given resource.

**IRI:** http://www.w3.org/ns/oa#textDirection

**Range:** [oa:Direction](http://www.w3.org/ns/oa#Direction)

**EXAMPLE 45:**

```xml
<http://example.org/anno44> a oa:Annotation;
    oa:hasBody [ rdf:value "This is a comment" ;
    dc:language "en" ;
    dc:format "text/plain" ;
    oa:textDirection oa:ltr ] ;
    oa:hasTarget <http://example.org/page1> .
```
2.2.29 via

A object of the relationship is a resource from which the source resource was retrieved by the providing system.

**IRI:** http://www.w3.org/ns/oa#via

---

**EXAMPLE 46: oa:via**

```xml
<http://example.org/anno45> a oa:Annotation ;
    oa:hasBody <http://example.org/note1> ;
    oa:hasTarget <http://example.org/page1> ;
    oa:via <http://other.example.com/anno1b> .
```

2.3 Named Individuals

2.3.1 assessing

The motivation for when the user intends to provide an assessment about the Target resource.

**IRI:** http://www.w3.org/ns/oa#assessing

**Instance Of:** oa:Motivation
2.3.2 bookmarking

The motivation for when the user intends to create a bookmark to the Target or part thereof.

IRI: http://www.w3.org/ns/oa#bookmarking

Instance Of: oa:Motivation
2.3.3 classifying

The motivation for when the user intends to classify the Target as something.

**IRI:** http://www.w3.org/ns/oa#classifying

**Instance Of:** oa:Motivation

---

**EXAMPLE 48: oa:bookmarking**

```html
<http://example.org/anno47> a oa:Annotation ;
    oa:hasTarget <http://example.com/page1> ;
    oa:motivatedBy oa:bookmarking .
```
2.3.4 commenting

The motivation for when the user intends to comment about the Target.

**IRI:** http://www.w3.org/ns/oa#commenting

**Instance Of:** oa:Motivation
2.3.5 describing

The motivation for when the user intends to describe the Target, as opposed to a comment about them.

**IRI:** http://www.w3.org/ns/oa#describing

**Instance Of:** oa:Motivation
2.3.6 editing

The motivation for when the user intends to request a change or edit to the Target resource.

**IRI:** http://www.w3.org/ns/oa#editing

**Instance Of:** oa:Motivation
2.3.7 highlighting

The motivation for when the user intends to highlight the Target resource or segment of it.

IRI: http://www.w3.org/ns/oa#highlighting

Instance Of: oa:Motivation
2.3.8 identifying

The motivation for when the user intends to assign an identity to the Target or identify what is being depicted or described in the Target.

IRI: http://www.w3.org/ns/oa#identifying

Instance Of: oa:Motivation
2.3.9 linking

The motivation for when the user intends to link to a resource related to the Target.

**IRI:** http://www.w3.org/ns/oa#linking

**Instance Of:** oa:Motivation

---

```xml
<http://example.org/anno53> a oa:Annotation ;
  oa:hasBody <http://example.com/identities/object1> ;
  oa:hasTarget <http://example.com/image-of-object1> ;
  oa:motivatedBy oa:identifying .
```
2.3.10 moderating

The motivation for when the user intends to assign some value or quality to the Target.

IRI: http://www.w3.org/ns/oa#moderating

Instance Of: oa:Motivation

EXAMPLE 55: oa:linking

\[
\text{<http://example.org/anno54> a } \text{oa:Annotation} ;
\quad \text{oa:hasBody}\ <\text{http://example.org/from1}> ;
\quad \text{oa:hasTarget}\ <\text{http://example.com/to1}> ;
\quad \text{oa:motivatedBy}\ \text{oa:linking}.
\]

EXAMPLE 56: oa:moderating

\[
\text{<http://example.org/anno55> a } \text{oa:Annotation} ;
\quad \text{oa:hasBody}\ <\text{http://example.org/tags/approved1}> ;
\quad \text{oa:hasTarget}\ <\text{http://example.com/anno1}> ;
\quad \text{oa:motivatedBy}\ \text{oa:moderating}.
\]

2.3.11 questioning
The motivation for when the user intends to ask a question about the Target.

**IRI:** http://www.w3.org/ns/oa#questioning

**Instance Of:** oa:Motivation

---

**EXAMPLE 57:** oa:questioning

```
<http://example.org/anno56> a oa:Annotation ;
  oa:hasBody [ a oa:TextualBody ;
    rdf:value "A question about the resource" ] ;
  oa:hasTarget <http://example.com/resource1> ;
  oa:motivatedBy oa:questioning .
```

---

### 2.3.12 replying

The motivation for when the user intends to reply to a previous statement, either an Annotation or another resource.

**IRI:** http://www.w3.org/ns/oa#replying

**Instance Of:** oa:Motivation
EXAMPLE 58: oa:replying

```xml
<http://example.org/anno57> a oa:Annotation ;
   oa:hasBody [ 
      a oa:TextualBody ;
      rdf:value "A reply to a question" ] ;
   oa:hasTarget <http://example.com/anno1> ;
   oa:motivatedBy oa:replying .
```

### 2.3.13 tagging

The motivation for when the user intends to associate a tag with the Target.

**IRI:** http://www.w3.org/ns/oa#tagging

**Instance Of:** [oa:Motivation](#)
2.3.14 autoDirection

The direction of text that should be automatically determined from the content.

**IRI:** http://www.w3.org/ns/oa#autoDirection

**Instance Of:** oa:Direction
2.3.15 ltrDirection

The direction of text that is read from left to right.

**IRI:** http://www.w3.org/ns/oa#ltrDirection

**Instance Of:** oa:Direction

**EXAMPLE 60: oa:autoDirection**

```xml
<http://example.org/anno59> a oa:Annotation ;
   oa:hasBody [ a oa:TextualBody ;
                 rdf:value "Some text" ;
                 oa:textDirection oa:autoDirection ] ;
   oa:hasTarget <http://example.com/thing1> .
```

---

2.3.15 ltrDirection

The direction of text that is read from left to right.

**IRI:** http://www.w3.org/ns/oa#ltrDirection

**Instance Of:** oa:Direction
2.3.16 rtlDirection

The direction of text that is read from right to left.

IRI: http://www.w3.org/ns/oa#rtlDirection

Instance Of: oa:Direction
2.3.17 PreferContainedDescriptions

An IRI to signal the client prefers to receive full descriptions of the Annotations from a container, not just their IRIs.

IRI: http://www.w3.org/ns/oa#PreferContainedDescriptions

Instance Of: rdfs:Resource
2.3.18 PreferContainedIRIs

An IRI to signal that the client prefers to receive only the IRIs of the Annotations from a container, not their full descriptions.

IRI: http://www.w3.org/ns/oa#PreferContainedIRIs
Instance Of: rdfs:Resource

3. Recommended Ontologies

3.1 Classes

3.1.1 as:Application

IRI: http://www.w3.org/ns/activitystreams#Application

Equivalent Classes: schema:SoftwareApplication

Range Of: as:generator, dcterms:creator

EXAMPLE 65: as:Application

```
<http://example.org/anno62> a oa:Annotation ;
   oa:hasBody <http://example.net/review1> ;
   oa:hasTarget <http://example.com/restaurant1> ;
   as:generator <http://example.org/client1> .

<http://example.org/client1> a as:Application ;
   foaf:homepage <http://example.com/homepage1> ;
   foaf:name "Code v2.1" .
```
3.1.2 as:OrderedCollection

IRI: http://www.w3.org/ns/activitystreams#OrderedCollection

Range Of: as:partOf

Required Predicates: as:first

Recommended Predicates: as:.last, as:totalItems, rdfs:label

EXAMPLE 66: as:OrderedCollection

<http://example.org/collection1> a as:OrderedCollection ;
  rdfs:label "Example Collection" ;
  as:totalItems 42023 ;
  as:first <http://example.org/collection1/page1> ;
  as:last <http://example.org/collection1/page42> .

3.1.3 as:OrderedCollectionPage

IRI: http://www.w3.org/ns/activitystreams#OrderedCollectionPage

Range Of: as:first, as:last, as:next, as:prev

Required Predicates: as:next, as:items
Recommended Predicates: as:prev, as:startIndex, as:partOf

EXAMPLE 67: as:OrderedCollectionPage

<http://example.org/collection1/page1> a as:OrderedCollectionPage ;
    as:partOf <http://example.org/collection1> ;
    as:next <http://example.org/collection1/page2> ;
    as:startIndex 0 ;
    as:items (  
        <http://example.org/anno1>
        <http://example.org/anno2>
        <http://example.org/anno3> ) .

3.1.4 dctypes:Dataset

IRI: http://purl.org/dc/dcmitype/Dataset

Equivalent Classes: schema:Dataset
EXAMPLE 68: dctypes:Dataset

```xml
<http://example.org/anno63> a oa:Annotation ;
    oa:motivatedBy oa:commenting ;
    oa:hasBody <http://example.com/note1> ;
    oa:hasTarget <http://example.com/dataset1> .

<http://example.com/dataset1> a dctypes:Dataset ;
    dc:format "text/csv" .
```

3.1.5 dctypes:MovingImage

IRI: http://purl.org/dc/dcmitype/MovingImage

Equivalent Classes: schema:VideoObject
Example 69: dcterms:MovingImage

```
<http://example.org/anno64> a oa:Annotation ;
    oa:hasBody [;
        a oa:TextualBody ;
        rdf:value "tag" ;
        oa:hasTarget <http://example.com/video1> ;
        oa:motivatedBy oa:tagging .

<http://example.org/video1> a dcterms:MovingImage .
```

3.1.6 dcterms:StillImage

**IRI:** http://purl.org/dc/dcmitype/StillImage

**Equivalent Classes:** schema:ImageObject
EXAMPLE 70: dctypes:StillImage

<http://example.org/anno65> a oa:Annotation ;
  oa:hasBody [
    a oa:TextualBody ;
    rdf:value "tag" ] ;
  oa:hasTarget <http://example.com/image1> ;
  oa:motivatedBy oa:tagging .

<http://example.org/image1> a dctypes:StillImage .
EXAMPLE 71: dctypes:Sound

```
<http://example.org/anno66> a oa:Annotation ;
    oa:hasBody [ a oa:TextualBody ;
    rdf:value "tag" ] ;
    oa:hasTarget <http://example.com/audio1> ;
    oa:motivatedBy oa:tagging .

<http://example.org/audio1> a dctypes:Sound .
```

3.1.8 dctypes:Text

IRI: http://purl.org/dc/dcmitype/Text
EXAMPLE 72: dctypes:Text

```xml
<http://example.org/anno67> a oa:Annotation ;
   oa:hasBody [
      a oa:TextualBody ;
      rdf:value "tag" ] ;
   oa:hasTarget <http://example.com/document1> ;
   oa:motivatedBy oa:tagging .

<http://example.org/document1> a dctypes:Text .
```

3.1.9 foaf:Organization

IRI: http://xmlns.com/foaf/0.1/Organization

Equivalent Classes: schema:Organization, as:Organization
EXAMPLE 73: foaf:Organization

<http://example.org/anno68> a oa:Annotation ;
  oa:hasBody <http://example.net/comment1> ;
  oa:hasTarget <http://example.com/restaurant1> ;
  dcterms:creator <http://example.org/org1> .

<http://example.org/org1> a foaf:Organization ;
  foaf:name "Example Organization" .

3.1.10 foaf:Person

IRI: http://xmlns.com/foaf/0.1/Person

Equivalent Classes: schema:Person, as:Person
EXAMPLE 74: foaf:Person

<http://example.org/anno69> a oa:Annotation ;
  oa:hasBody <http://example.net/review1> ;
  oa:hasTarget <http://example.com/book1> ;
  dcterms:creator <http://example.org/user1> .

<http://example.org/user1> a foaf:Person ;
  foaf:nick "pseudo" ;
  foaf:name "My Pseudonym" .

3.1.11 schema:Audience

IRI: http://schema.org/Audience

Range Of: schema:audience
EXAMPLE 75: schema:Audience

```
<http://example.org/anno70> a owl:Annotation ;
  owl:hasBody <http://example.net/review1> ;
  owl:hasTarget <http://example.com/book1> ;
  schema:audience <http://example.net/roles/musician> .

<http://example.net/roles/musician> a schema:Audience ;
  schema:audienceType "musician" .
```

3.2 Properties
### 3.2.1 as:first

**IRI:** [http://www.w3.org/ns/activitystreams#first](http://www.w3.org/ns/activitystreams#first)

**Domain:** `as:OrderedCollection`

**Range:** `as:OrderedCollectionPage`

---

**EXAMPLE 76: as:first**

```
<http://example.org/collection1> a as:OrderedCollection ;
  rdfs:label "Example Collection" ;
  as:totalItems 42023 ;
  as:first <http://example.org/collection1/page1> ;
  as:last <http://example.org/collection1/page42> .
```

---

### 3.2.2 as:generator

**IRI:** [http://www.w3.org/ns/activitystreams#generator](http://www.w3.org/ns/activitystreams#generator)
EXAMPLE 77: as:generator

```xml
<http://example.org/anno71> a oa:Annotation ;
    oa:hasBody <http://example.net/review1> ;
    oa:hasTarget <http://example.com/restaurant1> ;
    as:generator <http://example.org/client1> .

<http://example.org/client1> a as:Application ;
    foaf:homepage <http://example.com/homepage1> ;
    foaf:name "Code v2.1" .
```

3.2.3 as:items

<table>
<thead>
<tr>
<th>IRI</th>
<th><a href="http://www.w3.org/ns/activitystreams#items">http://www.w3.org/ns/activitystreams#items</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>as:OrderedCollectionPage</td>
</tr>
<tr>
<td>Range</td>
<td>rdf:List</td>
</tr>
</tbody>
</table>
EXAMPLE 78: as:items

```html
<http://example.org/collection1/page1> a as:OrderedCollectionPage ;
  as:partOf <http://example.org/collection1> ;
  as:next <http://example.org/collection1/page2> ;
  as:startIndex 0 ;
  as:items ( 
  <http://example.org/anno1>
  <http://example.org/anno2>
  # ...
  <http://example.org/anno1000> ) .
```

3.2.4 as:last

IRI: [http://www.w3.org/ns/activitystreams#last](http://www.w3.org/ns/activitystreams#last)

Domain: as:OrderedCollection

Range: as:OrderedCollectionPage
EXAMPLE 79: as:last

```xml
<http://example.org/collection1> a as:OrderedCollection ;
    rdfs:label "Example Collection" ;
    as:totalItems 42023 ;
    as:first <http://example.org/collection1/page1> ;
    as:last <http://example.org/collection1/page42> .
```

3.2.5 as:next

<table>
<thead>
<tr>
<th>IRI:</th>
<th><a href="http://www.w3.org/ns/activitystreams#next">http://www.w3.org/ns/activitystreams#next</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain:</td>
<td>as:OrderedCollectionPage</td>
</tr>
<tr>
<td>Range:</td>
<td>as:OrderedCollectionPage</td>
</tr>
</tbody>
</table>
EXAMPLE 80: as:next

```rdf
<http://example.org/collection1/page1> a as:OrderedCollectionPage ;
    as:partOf <http://example.org/collection1> ;
    as:next <http://example.org/collection1/page2> ;
    as:startIndex 0 ;
    as:items ( 
        <http://example.org/anno1>
        <http://example.org/anno2>
        # ...
        <http://example.org/anno1000> ) .
```

### 3.2.6 as:partOf

- **IRI**: [http://www.w3.org/ns/activitystreams#partOf](http://www.w3.org/ns/activitystreams#partOf)
- **Domain**: as:OrderedCollectionPage
- **Range**: as:OrderedCollection
3.2.7 as:prev

IRI: http://www.w3.org/ns/activitystreams#prev

Domain: as:OrderedCollectionPage

Range: as:OrderedCollectionPage

EXAMPLE 81: as:partOf

<http://example.org/collection1/page1> a as:OrderedCollectionPage ;
   as:partOf <http://example.org/collection1> ;
   as:next <http://example.org/collection1/page2> ;
   as:startIndex 0 ;
   as:items ( <http://example.org/anno1>
               <http://example.org/anno2>
               # ...
               <http://example.org/anno1000> ) .
EXAMPLE 82: as:prev

```xml
<http://example.org/collection1/page2> a as:OrderedCollectionPage ;
   as:partOf <http://example.org/collection1> ;
   as:prev <http://example.org/collection1/page1> ;
   as:next <http://example.org/collection1/page3> ;
   as:startIndex 1000 ;
   as:items (  
       <http://example.org/anno1001>  
       <http://example.org/anno1002>  
       # ...  
```

3.2.8 as:startIndex

IRI: http://www.w3.org/ns/activitystreams#startIndex

Domain: as:OrderedCollectionPage

Range: xsd:nonNegativeInteger
3.2.9 as:totalItems

IRI: http://www.w3.org/ns/activitystreams#totalItems

Domain: as:OrderedCollection

Range: xsd:nonNegativeInteger
EXAMPLE 84: as:totalItems

```xml
<http://example.org/collection1> a as:OrderedCollection ;
    rdfs:label "Example Collection" ;
    as:totalItems 42023 ;
    as:first <http://example.org/collection1/page1> ;
    as:last <http://example.org/collection1/page42> .
```

3.2.10 dc:format

| IRI:    | http://purl.org/dc/elements/1.1/format |
| Range:  | xsd:string                           |
3.2.11 dc:language

The dc:language predicate recommends the use of a controlled vocabulary. The use of dc:language in the Annotation model further specifies that the vocabulary to use is [BCP47].

IRI: http://purl.org/dc/elements/1.1/language

Range: xsd:string
EXAMPLE 86: dc:language

```html
<http://example.org/anno73> a oa:Annotation ;
   oa:hasTarget <http://example.org/photo1> ;
   oa:hasBody [a oa:TextualBody ;
                rdf:value "<p>Comment text</p>" ;
                dc:language "en" ;
```

3.2.12 dcterms:conformsTo

**IRI:** http://purl.org/dc/terms/conformsTo
EXAMPLE 87: dcterms:conformsTo

```
<http://example.org/anno74> a oa:Annotation ;
    oa:hasBody <http://example.org/image1> ;
    oa:hasTarget [ 
        oa:hasSource <http://example.org/video1> ;
        oa:hasSelector [ 
            a oa:FragmentSelector ; 
            dcterms:conformsTo <http://www.w3.org/TR/media-frags/> ;
            rdf:value "t=30,60" ] ] .
```
3.2.14 dcterms:creator

**IRI:** [http://purl.org/dc/terms/creator](http://purl.org/dc/terms/creator)

**Equivalent Properties:** as:actor
3.2.15 dcterms:issued

IRI: http://purl.org/dc/terms/issued

Range: xsd:dateTime

EXAMPLE 89: dcterms:creator

<http://example.org/anno76> a oa:Annotation ;
  oa:hasBody <http://example.net/post1> ;
  oa:hasTarget <http://example.com/page1> ;
  dcterms:creator <http://example.org/person1> .
EXAMPLE 90: dcterms:issued

```
<http://example.org/anno77> a oa:Annotation ;
   oa:hasBody <http://example.net/review1> ;
   oa:hasTarget <http://example.com/restaurant1> ;
   dcterms:issued "2015-10-14T15:13:28Z" ;
   as:generator <http://example.org/client1> .

<http://example.org/client1> a as:Application .
```

3.2.16 dcterms:modified

IRI: http://purl.org/dc/terms/modified

Range: xsd:dateTime
EXAMPLE 91: dcterms:modified

```xml
<http://example.org/anno78> a oa:Annotation ;
  oa:hasBody <http://example.net/review1> ;
  oa:hasTarget <http://example.com/restaurant1> ;
  dcterms:created "2015-10-13T13:00:00Z" ;
  dcterms:modified "2015-11-29T10:00:00Z" .
```

3.2.17 dcterms:rights

**IRI:** [http://purl.org/dc/terms/rights](http://purl.org/dc/terms/rights)
3.2.18 foaf:homepage

IRI: http://xmlns.com/foaf/0.1/homepage
EXAMPLE 93: foaf:homepage

```xml
<http://example.org/anno80> a oa:Annotation ;
oa:hasBody  <http://example.net/review1> ;
oa:hasTarget <http://example.com/restaurant1> ;
as:generator <http://example.org/client1> .

<http://example.org/client1> a as:Application ;
foaf:homepage <HomePage1> ;
foaf:name "Code v2.1" .
```

3.2.19 foaf:mbox

**IRI:** http://xmlns.com/foaf/0.1/mbox
EXAMPLE 94: foaf:mbox

<http://example.org/anno81> a oa:Annotation ;
  oa:hasBody <http://example.net/review1> ;
  oa:hasTarget <http://example.com/book1> ;
  dcterms:creator <http://example.org/user1> .

<http://example.org/user1> a foaf:Person ;
  foaf:mbox <mailto:user1@example.org> .

3.2.20 foaf:mbox_sha1sum

IRI: http://xmlns.com/foaf/0.1/mbox_sha1sum
EXAMPLE 95: foaf:mbox_sha1sum

<http://example.org/anno82> a oa:Annotation ;
   oa:hasBody <http://example.net/review1> ;
   oa:hasTarget <http://example.com/book1> ;
   dcterms:creator <http://example.org/user1> .

<http://example.org/user1> a foaf:Person ;
   foaf:mbox_sha1sum "7d97e98af710c7e7fe703abc8f639e0ee507c4" .

3.2.21 foaf:name

IRI: http://xmlns.com/foaf/0.1/name
EXAMPLE 96: foaf:name

<http://example.org/anno83> a oa:Annotation ;
  oa:hasBody <http://example.net/review1> ;
  oa:hasTarget <http://example.com/book1> ;
  dcterms:creator <http://example.org/user1> .

<http://example.org/user1> a foaf:Person ;
  foaf:nick "pseudo" ;
  foaf:name "My Pseudonym" .

3.2.22 foaf:nick

IRI: http://xmlns.com/foaf/0.1/nick
EXAMPLE 97: foaf:nick

```
<http://example.org/anno84> a oa:Annotation ;
    oa:hasBody <http://example.net/review1> ;
    oa:hasTarget <http://example.com/book1> ;
    dcterms:creator <http://example.org/user1> .

<http://example.org/user1> a foaf:Person ;
    foaf:nick "pseudo" ;
    foaf:name "My Pseudonym" .
```
EXAMPLE 98: rdf:type

```
<http://example.org/anno85> a oa:Annotation ;
  oa:hasBody <http://example.org/post1> ;
  oa:hasTarget <http://example.com/page1> ;
  oa:motivatedBy oa:commenting ;
  dcterms:creator <http://example.org/person1> ;
  dcterms:created "2015-11-18T12:00:00Z" .
```

3.2.24 rdf:value

**IRI:** [http://www.w3.org/1999/02/22-rdf-syntax-ns#value](http://www.w3.org/1999/02/22-rdf-syntax-ns#value)
EXAMPLE 99: rdf:value

<http://example.org/anno86> a oa:Annotation ;
oa:hasTarget <http://example.org/photo1> ;
oa:hasBody [ a oa:TextualBody;
  rdf:value "<p>Comment text</p>" ;
  dc:language "en" ;

3.2.25 rdfs:label

IRI: http://www.w3.org/2000/01/rdf-schema#label

Equivalent Properties: as:name
EXAMPLE 100: rdfs:label

```
<http://example.org/collection1> a as:OrderedCollection ;
    rdfs:label "Example Collection" ;
    as:totalItems 42023 ;
    as:first <http://example.org/collection1/page1> ;
    as:last <http://example.org/collection1/page42> .
```

3.2.26 schema:accessibilityFeature

IRI: http://schema.org/accessibilityFeature
EXAMPLE 101: schema:accessibilityFeature

```xml
<http://example.org/anno87> a oa:Annotation ;
  oa:hasBody <http://example.net/review1> ;
  oa:hasTarget <http://example.com/video1> .

<http://example.com/video1> a dctypes:MovingImage ;
  schema:accessibilityFeature "captions" .
```

3.2.27 schema:audience

IRI: http://schema.org/audience
4. Extensions

This vocabulary may be extended in the regular way, by creating new or importing existing predicates and classes from other RDF based ontologies. Extensions may be made to any resource, including adding new objects as well as properties. If there is a property in one of the ontologies that are already included in the context, then it is recommended to use a CURIE, the namespace and property name separated by a `:` character, as the JSON-LD key rather than creating a new context.

For example, it is easier to add `skos:prefLabel` to the Annotation rather than requiring clients to download a new context document to discover the mapping.
In order to ensure a lack of collision between key names, a JSON-LD context document should be made available when the term is not from an ontology that is already in the Web Annotation context. Extension contexts must not redefine existing JSON-LD keys from the Web Annotation context. Implementations must ignore unfamiliar properties when processing the data, but servers should preserve them if they are part of a valid and included context.

The context should be associated with the resource that most closely encapsulates the extension properties. This is to try to ensure that extensions which define the same key do not collide unexpectedly. If there is more than one context document for a particular resource, then they must be included in an array.

For example, adding height and width for the target image using the EXIF vocabulary [exif] could be done by defining a JSON-LD context, including the height and width properties, and linking to the newly defined context in the target resource.
Note that the current JSON-LD [JSON-LD] specification does not allow arrays to contain other arrays directly. As JSON-LD is the recommended serialization format, extensions should avoid the use of this pattern.

Finally, new classes can be defined to further extend the model for specific communities and use cases. New Selectors and States are particularly valuable for extension for new ways of defining representations, and for selecting segments of those representations.

For example, an annotation on a three dimensional model would require the addition of depth and the starting position on the z axis, along with x, y, width, and height. These might be kept together in a new ThreeDSelector resource.
A. JSON-LD Context

This section is non-normative.

The recommended serialization format is [JSON-LD]. The JSON-LD context presented below is recommended to ensure consistency between implementations, and should be referenced as http://www.w3.org/ns/anno.jsonld. The same IRI should be used as the profile IRI for representations that conform to the model and context.

```json
EXAMPLE 105: Extension Example 3

{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/anno91",
    "type": "Annotation",
    "skos:prefLabel": "3d Annotation",
    "body": {
        "type": "TextualBody",
        "value": "I love this part of the model!"
    },
    "target": {
        "source": "http://example.org/models/robot.3d",
        "selector": {
            "@context": "http://example.org/3d/ns/extension.jsonld",
            "type": "ThreeDSelector",
            "x": 1035,
            "y": 245,
            "z": 782,
            "w": 120,
            "h": 180,
            "d": 90
        }
    }
}
```
{
"@context": {
"oa":
"dc":
"dcterms":
"dctypes":
"foaf":
"rdf":
"rdfs":
"skos":
"xsd":
"iana":
"owl":
"as":
"schema":
"id":
"type":

"http://www.w3.org/ns/oa#",
"http://purl.org/dc/elements/1.1/",
"http://purl.org/dc/terms/",
"http://purl.org/dc/dcmitype/",
"http://xmlns.com/foaf/0.1/",
"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
"http://www.w3.org/2000/01/rdf-schema#",
"http://www.w3.org/2004/02/skos/core#",
"http://www.w3.org/2001/XMLSchema#",
"http://www.iana.org/assignments/relation/",
"http://www.w3.org/2002/07/owl#",
"http://www.w3.org/ns/activitystreams#",
"http://schema.org/",
{"@type": "@id", "@id": "@id"},
{"@type": "@id", "@id": "@type"},

"Annotation":
"Dataset":
"Image":
"Video":
"Audio":
"Text":
"TextualBody":
"ResourceSelection":
"SpecificResource":
"FragmentSelector":
"CssSelector":
"XPathSelector":
"TextQuoteSelector":
"TextPositionSelector":
"DataPositionSelector":
"SvgSelector":
"RangeSelector":
"TimeState":
"HttpRequestState":
"CssStylesheet":
"Choice":
"Person":
"Software":
"Organization":
"AnnotationCollection":

"oa:Annotation",
"dctypes:Dataset",
"dctypes:StillImage",
"dctypes:MovingImage",
"dctypes:Sound",
"dctypes:Text",
"oa:TextualBody",
"oa:ResourceSelection",
"oa:SpecificResource",
"oa:FragmentSelector",
"oa:CssSelector",
"oa:XPathSelector",
"oa:TextQuoteSelector",
"oa:TextPositionSelector",
"oa:DataPositionSelector",
"oa:SvgSelector",
"oa:RangeSelector",
"oa:TimeState",
"oa:HttpRequestState",
"oa:CssStyle",
"oa:Choice",
"foaf:Person",
"as:Application",
"foaf:Organization",
"as:OrderedCollection",


null
B. JSON-LD Frames

This section is non-normative.
There is an unofficial, yet well implemented, JSON-LD specification [json-ld-framing] that describes a deterministic layout for serializing an RDF graph into a particular JSON-LD document layout. Applying the following frames to the graph of information will generate JSON as close as possible to the serialization recommended by the Web Annotation Data Model.

B.1 Annotation Frame

A Frame for serializing a single Annotation.

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "@omitDefault": true,
  "type": "Annotation",
  "via": {"@embed": false},
  "canonical": {"@embed": false},
  "rights": {"@embed": false},
  "motivation": {"@embed": false},
  "body": {"@embed": true},
  "target": {"@embed": true},
  "creator": {"@embed": true},
  "generator": {"@embed": true},
  "audience": {"@embed": true}
}
```

B.2 Annotation Collection Frame

A Frame for serializing a Collection of Annotations.

```
{
  "@context" : [
    "http://www.w3.org/ns/anno.jsonld",
    "http://www.w3.org/ns/ldp.jsonld"
  ],
  "type": "Collection",
  "first": [{"@embed": "False"}],
  "last": [{"@embed": "False"}]
}
```
NOTE
If it is instead desirable to embed the first page of the Collection, change the `false` for `first` to `true`.

B.3 Annotation Page Frame

A Frame for serializing a Page from a Collection of Annotations.

```json
{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "@omitDefault": true,
    "type": "AnnotationPage",
    "next": {"@embed": false},
    "prev": {"@embed": false},
    "partOf": {"@embed": false},
    "items": {"@embed": true}
}
```

C. Extending Motivations

This section is non-normative.

Although the list of Motivations in the specification is derived from an extensive survey of the annotation landscape, there are many situations where more precise definitions are required or desirable. In these cases it is recommended to create a new Motivation resource and relate it to one or more that already exist.

New Motivations must be instances of `oa:Motivation`, which is a subClass of `skos:Concept`. The `skos:broader` relationship should be asserted between the new Motivation and at least one existing Motivation, if there are any that are broader in scope. Other relationships, such as `skos:relatedMatch`, `skos:exactMatch` and `skos:closeMatch`, should also be asserted to concepts created by other communities.

Model
D. Proposed Definitions

This section is non-normative.

The following classes were proposed for the ontology, but did not have the necessary implementations and were consequently removed.

D.1 Composite

A subClass of as:OrderedCollection that conveys to a consuming application that it should use all of the resources in the as:items list, but that order is not important.

IRI: http://www.w3.org/ns/oa#Composite

Sub Class Of: as:OrderedCollection

Required Predicates: as:items, rdf:type
D.2 Independents

A subClass of as:OrderedCollection that conveys to a consuming application that each of the resources in the as:items list are independently associated with all of the other bodies or targets.

IRI: http://www.w3.org/ns/oa#Independents

Sub Class Of: as:OrderedCollection

Required Predicates: as:items, rdf:type

EXAMPLE 106: oa:Composite

<http://example.org/anno92> a oa:Annotation ;
  oa:motivatedBy oa:commenting ;
  oa:hasBody [ 
    a oa:TextualBody ;
    rdf:value "These pages together provide evidence of the conspiracy";
  ]
  oa:hasTarget [ 
    a oa:Composite ;
    as:items ( <http://example.org/page1> <http://example.org/page6> ... )
  ]
D.3 List

A subClass of `as:OrderedCollection` that conveys to a consuming application that it should use each of the resources in the `as:items` list, and that their order is important.

**IRI:** [http://www.w3.org/ns/oa#Independents](http://www.w3.org/ns/oa#Independents)

**Sub Class Of:** `as:OrderedCollection`
EXAMPLE 108: oa:List

<http://example.org/anno94> a oa:Annotation ;
   oa:motivatedBy oa:tagging ;
   oa:hasBody [ a oa:TextualBody ;
             rdf:value "important" ] ;
   oa:hasTarget [ a oa:List ;
                 as:items ( 
                   <http://example.com/book/page1>
                   <http://example.net/book/page2>
                   <http://example.com/book/page3>

E. Candidate Recommendation Exit Criteria

This section is non-normative.
For this specification to be advanced to Proposed Recommendation, there must be at least two independent implementations that demonstrate the validity of the vocabulary. The vocabulary will be considered valid when the following conditions have been demonstrated:

- The JSON-LD context document can be parsed without errors by JSON-LD validators.
- The JSON-LD context document can be used to convert JSON-LD serialized Annotations into RDF triples.
- The graphs produced by different implementations are isomorphic.
- The graphs produced can be converted back into JSON-LD without loss of information, within the limits of the existing standards and excluding extensions to the vocabulary.
- The ontology documents can be parsed without errors by RDF Schema validators.
- The ontology is internally consistent with respect to domains, ranges, inverses, and any other ontology features specified.

NOTE

The recommended pattern for extensions in JSON-LD of including the extension context document in the serialization of the resource that uses the extension predicates and/or classes is not able to be round-tripped, as JSON-LD does not maintain the alignment between context and resource once transformed into a graph. The order and exact naming of the JSON keys in the JSON-LD serialization resulting from a graph is not considered a feature of the Vocabulary, and thus not required for demonstrating its validity. The naming of the keys is instead a feature of the Annotation Model and verified by its exit criteria.

F. Changed from Previous Versions

This section is non-normative.

F.1 Changes from the Proposed Recommendation of 2017-01-17

No significant changes.

F.2 Changes from the Candidate Recommendation of 2016-11-22
• Removed at-risk status of ActivityStreams predicates

F.3 Changes from the Candidate Recommendation of 2016-09-06

• Moved Independents, Composite and List to an informative appendix
• Updated JSON-LD context appendix
• Minor formatting issues
• Fixed appendices to be marked as non-normative by respec

F.4 Changes from the Candidate Recommendation of 2016-07-05

• Added CR Exit Criteria

F.5 Changes from the Working Draft of 2016-03-31

Significant technical changes in this specification from the Working Draft Published of 2016-03-31 are:

• Add new classes: Independents, Composite, List, Direction
• Remove Content class
• Add new individuals: autoDirection, ltrDirection, rtlDirection
• Rename oa:reviewing to broader oa:assessing
• Add Preference individuals from Protocol: PreferContainedDescriptions, PreferContainedIRIs
• Add new predicates: textDirection, processingLanguage
• Reference newly imported predicates: schema:accessibilityFeature
• Rename oa:bodyText to oa:bodyValue
• Use IRI instead of URI in the definitions
• Finish diagrams and examples

G. Acknowledgments

This section is non-normative.
The Web Annotation Working Group gratefully acknowledges the contributions of the Open Annotation Community Group. The output of the Community Group was fundamental to the current data model.

The following people have been instrumental in providing thoughts, feedback, reviews, content, criticism and input in the creation of this specification:


H. References

H.1 Normative references

[BCP47]

[DC-TERMS]

[DC11]
Dublin Core Metadata Element Set, Version 1.1. Dublin Core metadata initiative. 14 June 2012. DCMI recommendation. URL: http://dublincore.org/documents/2012/06/14/dces/

[FOAF]

[JSON-LD]
JSON-LD 1.0. Manu Sporny; Gregg Kellogg; Markus Lanthaler. W3C. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/json-ld/

[RFC2119]

[Turtle]

RDF 1.1 Turtle. Eric Prud’hommeaux; Gavin Carothers. W3C. 25 February 2014. W3C Recommendation. URL: https://www.w3.org/TR/turtle/

[activitystreams-core]

Activity Streams 2.0. James Snell; Evan Prodromou. W3C. 15 December 2016. W3C Candidate Recommendation. URL: https://www.w3.org/TR/activitystreams-core/

[activitystreams-vocabulary]


[annotation-model]

Web Annotation Data Model. Robert Sanderson; Paolo Ciccarese; Benjamin Young. W3C. W3C Recommendation. URL: http://www.w3.org/TR/annotation-model/

[annotation-protocol]


[rdf-schema]


[rfc3987]


[rfc5988]


[skos-reference]


[xmlschema-2]


H.2 Informative references

[exif]

Exif Vocabulary Workspace - RDF Schema. W3C. URL: https://www.w3.org/2003/12/exif/
JSON-LD Framing 1.0. W3C JSON-LD Community Group. URL: http://json-ld.org/spec/latest/json-ld-framing/

Web Annotation Protocol
W3C Recommendation 23 February 2017

This version:
https://www.w3.org/TR/2017/REC-annotation-protocol-20170223/

Latest published version:
https://www.w3.org/TR/annotation-protocol/

Latest editor's draft:
https://w3c.github.io/web-annotation/protocol/wd/

Implementation report:
https://w3c.github.io/test-results/annotation-protocol/all.html

Previous version:
https://www.w3.org/TR/2017/PR-annotation-protocol-20170117/

Editor:
Robert Sanderson, J Paul Getty Trust, rsanderson@getty.edu

Repository:
Github Repository

Changes:
Diff to previous version
Commit history

Please check the errata for any errors or issues reported since publication.

This document is also available in this non-normative format: ePub

The English version of this specification is the only normative version. Non-normative translations may also be available.

Copyright © 2017 W3C® (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

Abstract

Annotations are typically used to convey information about a resource or associations between resources. Simple examples include a comment or tag on a single web page or image, or a blog post about a news article.
The Web Annotation Protocol describes the transport mechanisms for creating and managing annotations in a method that is consistent with the Web Architecture and REST best practices.

**Status of This Document**

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/*.

By publishing this Recommendation, W3C expects that the functionality specified in this Recommendation will not be affected by changes to the Activity Streams 2.0 [activitystreams-core] and Activity Vocabulary [activitystreams-vocabulary] as those specifications proceed to Recommendation.

This document was published by the Web Annotation Working Group as a Recommendation. If you wish to make comments regarding this document, please send them to public-annotation@w3.org (subscribe, archives). All comments are welcome.

Please see the Working Group's implementation report.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C’s role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 September 2015 W3C Process Document.

**Table of Contents**

1. **Introduction**
   1.1 Aims of the Protocol
   1.2 Summary
   1.3 Conformance
1.4 Terminology

2. Web Annotation Protocol Principles

3. Annotation Retrieval

4. Annotation Containers
4.1 Container Retrieval
4.2 Container Representations
4.2.1 Container Representation Preferences
4.2.2 Representations without Annotations
4.2.3 Representations with Annotation IRIs
4.2.4 Representations with Annotation Descriptions
4.3 Annotation Pages
4.4 Discovery of Annotation Containers

5. Creation, Updating and Deletion of Annotations
5.1 Create a New Annotation
5.2 Suggesting an IRI for an Annotation
5.3 Update an Existing Annotation
5.4 Delete an Existing Annotation

6. Error Conditions

7. Containers for Related Resources

A. Candidate Recommendation Exit Criteria

B. Changes from Previous Versions
B.1 Changes from the Proposed Recommendation of 2017-01-17
B.2 Changes from the Candidate Recommendation of 2016-09-06
B.3 Changes from the Candidate Recommendation of 2016-07-12
B.4 Changes from the Working Draft of 2016-03-31

C. Acknowledgments

D. References
D.1 Normative references
1. Introduction

This section is non-normative.

Interoperability between systems has two basic aspects: the syntax and semantics of the data that is moved between the systems, and the transport mechanism for that movement. The HTTP protocol and the Web architecture provides us with a great starting point for a standardized transport layer, and can be used to move content between systems easily and effectively. Building upon these foundations allows us to make use of existing technology and patterns to ensure consistency and ease of development.

The Web Annotation Protocol describes a transport mechanism for creating, managing, and retrieving Annotations. Annotations in this specification are assumed to follow the requirements of the Web Annotation Data Model [annotation-model] and Web Annotation Vocabulary [annotation-vocab]. This specification builds upon REST principles and the Linked Data Platform [ldp] recommendation, and familiarity with it is recommended.

1.1 Aims of the Protocol

The primary aim of the Web Annotation Protocol is to provide a standard set of interactions that allow annotation clients and servers to interoperate seamlessly. By being able to discover annotation protocol end-points and how to interact with them, clients can be configured either automatically or by the user to store annotations in any compatible remote system, rather than being locked in to a single client and server pair.

1.2 Summary

For those familiar with the Web Annotation model, LDP, and REST, much of the Annotation Protocol will be very obvious. The following aspects are the most important new requirements.

- The media type to use for Annotations is:
  application/ld+json;profile="http://www.w3.org/ns/anno.jsonld"

- Annotation Containers are constrained by the set of constraints described in this specification, and thus the ldp:constrainedBy URL is http://www.w3.org/TR/annotation-protocol/

- The link header can refer from any resource to an Annotation Container using a rel type of:
  http://www.w3.org/ns/oa#annotationService
The response from a Container after creating an Annotation should include a representation of the Annotation, after any changes have been made to it, in the JSON-LD serialization.

Annotation Containers should only contain Annotations, and not other resources.

- Activity Streams Collection [activitystreams-core] model is used for paging, as in-page ordering is an important requirement.

1.3 Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words may, must, must not, recommended, should, and should not are to be interpreted as described in [RFC2119].

1.4 Terminology

**IRI**

An IRI, or Internationalized Resource Identifier, is an extension to the URI specification to allow characters from Unicode, whereas URIs must be made up of a subset of ASCII characters. There is a mapping algorithm for translating between IRIs and the equivalent encoded URI form. IRIs are defined by [rfc3987].

**Resource**

An item of interest that may be identified by an IRI.

**Web Server**

A program that accepts connections in order to service HTTP requests by sending HTTP responses.

**Annotation**

A web resource that follows the Web Annotation Data Model [annotation-model].

**Annotation Server**

A Web Server that also makes available and allows the management of Annotations via the protocol described in this document.

**Annotation Client**

A program that establishes connections to Annotation Servers for the purpose of retrieving and managing Annotations via the protocol described in this document.

**Annotation Container**

An LDP Container [ldp] used to manage Annotations.
2. Web Annotation Protocol Principles

The Web Annotation Protocol is defined using the following basic principles:

- The protocol is developed within the framework laid out by the Web Architecture.
- Interactions will follow the REST best practice guidelines when there is a resource being acted upon.
- Interactions are designed to take place over HTTP.
- Existing specifications and systems will be re-used whenever possible, constrained further when necessary, with invention of new specifications only as a last resort.
- Simplicity and ease of implementation are important design criteria, but ultimately subjective and less important than the above principles.

3. Annotation Retrieval

The Annotation Server must support the following HTTP methods on the Annotation's IRI:

- **GET** (retrieve the description of the Annotation),
- **HEAD** (retrieve the headers of the Annotation without an entity-body),
- **OPTIONS** (enable CORS pre-flight requests [cors]).

Servers should use HTTPS rather than HTTP for all interactions, including retrieval of Annotations.

Servers must support the JSON-LD representation using the Web Annotation profile. These responses must have a Content-Type header with the application/ld+json media type, and it should have the Web Annotation profile IRI of http://www.w3.org/ns/anno.jsonld in the profile parameter.

Servers should support a Turtle representation, and may support other formats. If more than one representation of the Annotation is available, then the server should support content negotiation. Content negotiation for different serializations is performed by including the desired media type in the HTTP Accept header of the request, however clients cannot assume that the server will honor their preferences [rfc7231].

Servers may support different JSON-LD profiles. Content negotiation for different JSON-LD profiles is performed by adding a profile parameter to the JSON-LD media type in a space separated, quoted list as part of the Accept header.
Servers should use the 200 HTTP status code when no errors occurred while processing the request to retrieve an Annotation, and may use 3XX HTTP status codes to redirect to a new location.

The response from the Annotation Server must have a Link header entry where the target IRI is `http://www.w3.org/ns/ldp#Resource` and the rel parameter value is `type`. The Annotation type of `http://www.w3.org/ns/oa#Annotation` may also be added with the same rel type. This is to let client systems know that the retrieved representation is a Resource and an Annotation, even if the client cannot process the representation's format.

For HEAD and GET requests, the response must have an ETag header with an entity reference value that implements the notion of entity tags from HTTP [rfc7232]. This value will be used by the client when sending update or delete requests.

The response must have an Allow header that lists the HTTP methods available for interacting with the Annotation [rfc7231].

For HEAD and GET requests, if the server supports content negotiation by format or JSON-LD profile, the response must have a Vary header with Accept in the value [rfc7231]. This is to ensure that caches understand that the representation changes based on the value of that request header.

Request:

```
EXAMPLE 1

GET /annotations/anno1 HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
```

Response:
4. Annotation Containers

If the Annotation Server supports the management of Annotations, including one or more of creating, updating, and deleting them, then the following section's requirements apply. The Annotation Protocol is a use of the Linked Data Platform [ldp] specification, with some additional constraints derived from the Web Annotation Data Model [annotation-model].

An Annotation Server must provide one or more Containers within which Annotations can be managed: an Annotation Container. An Annotation Container is at the same time both a Container [ldp] (a service for managing Annotations) and an OrderedCollection [activitystreams-core] (an ordered list of Annotations). It can have descriptive and technical information associated with it to allow clients to present it to a user in order to allow her to decide if it should be used or not. The classes, properties and representations for the Collection model are described in the Web Annotation Data Model, and the mappings to Activity Streams provided in the Web Annotation Vocabulary [annotation-vocab].

Annotation Containers should implement the LDP Basic Container specification, but may instead implement another type of Container, such as a Direct or Indirect Container, to fulfill business needs. The
URI of an Annotation Container must not have a query or fragment component, and the path component must end in a "/" character.

Implementations should use HTTPS rather than HTTP for all interactions with Annotation Containers.

The creation, management, and structure of Annotation Containers are beyond the scope of this specification. Please see the Linked Data Platform specification [ldp] for additional information.

4.1 Container Retrieval

The Annotation Server must support the following HTTP methods on the Annotation Container's IRI:

- **GET** (retrieve the description of the Container and the list of its contents, described below),
- **HEAD** (retrieve the headers of the Container without an entity-body),
- **OPTIONS** (enable CORS pre-flight requests [cors]).

When an HTTP GET request is issued against the Annotation Container, the server must return a description of the container. That description must be available in JSON-LD, should be available in Turtle, and may be available in other formats. The JSON-LD serialization of the Container's description should use both the LDP context (http://www.w3.org/ns/ldp.jsonld), and the Web Annotation's profile and context [annotation-model], unless the request would determine otherwise.

Servers should use the 200 HTTP status code if the request is successfully completed without errors and does not require redirection based on the client's preferences.

All supported methods for interacting with the Annotation Container should be advertised in the Allow header of the GET, HEAD and OPTIONS responses from the container's IRI. The Allow header may also be included on any other responses.

Annotation Containers must return a Link header [rfc5988] on all responses with the following components:

- It must advertise its type by including a link where the rel parameter value is type and the target IRI is the appropriate Container Type, such as http://www.w3.org/ns/ldp#BasicContainer for Basic Containers.
- It must advertise that it imposes Annotation protocol specific constraints by including a link where the target IRI is http://www.w3.org/TR/annotation-protocol/, and the rel parameter value is the IRI http://www.w3.org/ns/ldp#constrainedBy.
For HEAD and GET requests, responses from Annotation Containers must include an ETag header that implements the notion of entity tags from HTTP [rfc7232]. This value should be used by administrative clients when updating the container by including it in an If-Match request header in the same way as clients wanting to update an Annotation.

If the Accept header is absent from a GET request, then Annotation Servers must respond with a JSON-LD representation of the Annotation Container, however clients with a preference for JSON-LD should explicitly request it using an Accept request header.

If the server supports content negotiation by format or JSON-LD profile, the response to a HEAD or GET request from the Annotation Container must have a Vary header that includes Accept in the value to ensure that caches can determine that the representation will change based on the value of this header in requests.

Responses from Annotation Containers that support the use of the POST method to create Annotations should include an Accept-Post header on responses to GET, HEAD and OPTIONS requests. The value is a comma separated list of media-types that are acceptable for the client to send via POST [ldp].

EXAMPLE 3: Example Annotation Container Headers

```
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type"
Link: <http://www.w3.org/TR/annotation-protocol/>; rel="http://www.w3.org/ns/ldp#constrainedBy"
ETag: "0f6b5cd8dc1f754a1738a53b1da34f6b"
Vary: Accept
Allow: POST, GET, OPTIONS, HEAD
Accept-Post: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
```

4.2 Container Representations

As there are likely to be many Annotations in a single Container, the Annotation Protocol adopts the ActivityStreams collection paging mechanism for returning the contents of the Container. Each Collection Page contains an ordered list with a subset of the managed Annotations, such that if every page is traversed, a client can reconstruct the complete, ordered contents of the container/collection. The number of IRIs or Annotation descriptions included on each page is at the server’s discretion, and may be inconsistent between pages. The feature or features by which the Annotations are sorted are not explicit in the response.
The requirements for JSON-LD representation of Annotation Collections are defined in the Web Annotation Data Model, and are summarized here.

The Collection must have an IRI that identifies it, and must have at least the AnnotationCollection class (the name associated with OrderedCollection in the JSON-LD context) but may have other types as well, including the type of LDP Container used. It should have a human readable label, and may have other properties such as creator and created.

If there are greater than zero Annotations in the Container, the representation must either include a link to the first page of Annotations as the value of the first property, or include the representation of the first page embedded within the response. If there is more than one page of Annotations, then the representation should have a link to the last page using the last property.

The representation of the Container should include the total property with the total number of annotations in the Container. The Container should include the modified property with the most recent timestamp of when any of the annotations in the Container. This timestamp allows clients to detect when to re-cache data, even if there are the same number of annotations as the same number may have been added and deleted.

The IRI of the Container provided in the response should differentiate between whether the pages contain just the IRIs, or the full descriptions of the Annotations. It is recommended that this be done with a query parameter. The server may redirect the client to this IRI and deliver the response there, otherwise it must include a Content-Location header with the IRI as its value.

4.2.1 Container Representation Preferences

There are three preferences for Container requests that will govern the representation of the server's responses:

1. If the client prefers to only receive the Container description and no Annotations (either URI or full descriptions) embedded in the Container response, then it must include a Prefer request header with the value return=representation;include="http://www.w3.org/ns/ldp#PreferMinimalContainer".

2. If the client prefers to receive the Annotations only as IRI references, either embedded in the current Container response or future paged responses, then it must include a Prefer request header with the value return=representation;include="http://www.w3.org/ns/oa#PreferContainedIRIs".
3. If the client prefers to receive complete Annotation descriptions, either in the current Container response or future paged responses, then it must include a Prefer request header with the value `return=representation;include="http://www.w3.org/ns/oa#PreferContainedDescriptions"`.

The client may send multiple preferences as the value of the include parameter as defined by the Linked Data Platform [ldp]. However, the client must not include both the PreferContainedIRIs and PreferContainedDescriptions preferences on the same request, as the server cannot honor both at the same time. If the PreferMinimalContainer preference is given, then the server should not embed the Annotations or references to them, but should include a reference to the first and last Annotation Pages. Whether the pages are of IRI references or complete descriptions is governed by the use of PreferContainedIRIs and PreferContainedDescriptions respectively. If no preference is given by the client, the server should default to the PreferContainedDescriptions behavior. The server may ignore the client's preferences.

### 4.2.2 Representations without Annotations

If the client requests the minimal representation of an Annotation Container, the response must not include either the ldp:contains predicate nor embed the first page of Annotations within the response.

The linked pages should follow any PreferContainedDescriptions or PreferContainedIRIs preferences.

The server may return a representation without embedded Annotations, even if the PreferMinimalContainer preference is not supplied.

**Request:**

**EXAMPLE 4: Container Request without Annotations**

```
GET /annotations/ HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Prefer: return=representation;include="http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

**Response:**
4.2.3 Representations with Annotation IRIs

If the Server supports Container preferences, it must respond to PreferContainedIRIs with a response containing an AnnotationPage as the value of first with its items containing only the IRIs of the contained Annotations.

The linked pages should follow the PreferContainedIRIs preference.

The PreferContainedIRIs and the PreferContainedDescriptions preferences are mutually exclusive.

Request for embedded IRIs:
EXAMPLE 6: Container Request (Embedded IRIs)

GET /annotations/ HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Prefer: return=representation;include="http://www.w3.org/ns/oa#PreferCont

Response:
EXAMPLE 7: Container Response (Embedded IRIs)

HTTP/1.1 200 OK
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Location: http://example.org/annotations/?iris=1
ETag: "_87e52ce123123"
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type"
Link: <http://www.w3.org/TR/annotation-protocol/>; rel="http://www.w3.org/ns/ldp#constrainedBy"
Allow: POST, GET, OPTIONS, HEAD
Vary: Accept, Prefer
Content-Length: 397

{
   "@context": [
      "http://www.w3.org/ns/anno.jsonld",
      "http://www.w3.org/ns/ldp.jsonld"
   ],
   "id": "http://example.org/annotations/?iris=1",
   "type": ["BasicContainer", "AnnotationCollection"],
   "total": 42023,
   "modified": "2016-07-20T12:00:00Z",
   "label": "A Container for Web Annotations",
   "first": {
      "id": "http://example.org/annotations/?iris=1&page=0",
      "type": "AnnotationPage",
      "next": "http://example.org/annotations/?iris=1&page=1",
      "items": [
         "http://example.org/annotations/anno1",
         "http://example.org/annotations/anno2",
         "http://example.org/annotations/anno3",
         "http://example.org/annotations/anno4",
         "http://example.org/annotations/anno5",
         "http://example.org/annotations/anno6",
         ...
         "http://example.org/annotations/anno999",
      ]
   },
   "last": "http://example.org/annotations/?iris=1&page=42"
}

4.2.4 Representations with Annotation Descriptions
If the Server supports Container preferences, it must respond to `PreferContainedDescriptions` with a response containing an `AnnotationPage` as the value of `first` with its `items` containing complete, inline Annotations.

The linked pages should follow the `PreferContainedDescriptions` preference.

The `PreferContainedIRIs` and the `PreferContainedDescriptions` preferences are mutually exclusive.

Request for embedded descriptions:

```
EXAMPLE 8: Container Request (Embedded Descriptions)

GET /annotations/ HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Prefer: return=representation;include="http://www.w3.org/ns/oa#PreferCon...
```
EXAMPLE 9: Container Response (Embedded Descriptions)
HTTP/1.1 200 OK
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Allow: GET,OPTIONS,HEAD
Vary: Accept, Prefer
Content-Length: 924

{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://www.w3.org/ns/ldp.jsonld"
  ],
  "id": "http://example.org/annotations/?iris=0",
  "type": ["BasicContainer", "AnnotationCollection"],
  "total": 42023,
  "modified": "2016-07-20T12:00:00Z",
  "label": "A Container for Web Annotations",
  "first": {
    "id": "http://example.org/annotations/?iris=0&page=0",
    "type": "AnnotationPage",
    "next": "http://example.org/annotations/?iris=0&page=1",
    "items": [
      {
        "id": "http://example.org/annotations/anno1",
        "type": "Annotation",
        "body": "http://example.net/body1",
        "target": "http://example.com/page1"
      },
      {
        "id": "http://example.org/annotations/anno2",
        "type": "Annotation",
        "body": {
          "type": "TextualBody",
          "value": "I like this!"
        },
        "target": "http://example.com/book1"
      },
      ...
      {
        "id": "http://example.org/annotations/anno50",
        "type": "Annotation",
        "body": "http://example.org/texts/description1",
        "target": "http://example.com/images/image1"
      }
    ]
  }
}"
4.3 Annotation Pages

Individual pages are instances of the Activity Streams `OrderedCollectionPage` class, which is referred to as `AnnotationPage` in the Web Annotation JSON-LD context. The page contains the Annotations, either via their IRIs or full descriptions, in the `items` property.

The requirements for JSON-LD representation of Annotation Collections Pages are defined in the Web Annotation Data Model, and are summarized here.

The Annotation Page must have an IRI that identifies it, and must have at least the `AnnotationPage` class but may have other types as well. If the Page is not the last Page in the Collection, then it must have a reference to the Page which follows it using the `next` property. If the Page is not the first Page in the Collection, it should have a reference to the previous Page using the `prev` property. Pages should give the position of the first Annotation in the `items` list relative to the order of the Collection using the zero-based `startIndex` property.

Each page must have a link to the Collection that it is part of, using the `partOf` property. The description of the Collection should include both the `total` and `modified` properties. The response may include other properties of the Collection in the response, such as the `label` or `first` and `last` links.

The client should not send the `Prefer` header when requesting the Page, as it has already been taken into account when requesting the Collection.

This specification does not require any particular functionality when a client makes requests other than GET, HEAD or OPTIONS to a page.

As the Page is not an LDP Container, it does not have the requirement to include a `Link` header with a type. That the URLs could be constructed with query parameters added to the Container's IRI is an implementation convenience, and does not imply the type of the resource.

**Embedded IRIs Interaction Example**

Request:
EXAMPLE 10: Page Request

```
GET /annotations/?iris=1&page=0 HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
```

Response:

EXAMPLE 11: Page Response (Embedded IRIs)

```
HTTP/1.1 200 OK
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Allow: GET,OPTIONS,HEAD
Vary: Accept
Content-Length: 630

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/annotations/?iris=1&page=0",
  "type": "AnnotationPage",
  "partOf": {
    "id": "http://example.org/annotations/?iris=1",
    "total": 42023,
    "modified": "2016-07-20T12:00:00Z",
  },
  "startIndex": 0,
  "next": "http://example.org/annotations/?iris=1&page=1",
  "items": [
    "http://example.org/annotations/anno1",
    "http://example.org/annotations/anno2",
    "http://example.org/annotations/anno3",
    "http://example.org/annotations/anno4",
    "http://example.org/annotations/anno5",
    "http://example.org/annotations/anno6",
    ...
    "http://example.org/annotations/anno999",
  ]
}
```

Embedded Descriptions Interaction Example
Request:

EXAMPLE 12: Page Request (Embedded Descriptions)

GET /annotations/?iris=0&page=0 HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
EXAMPLE 13: Page Response (Embedded Descriptions)

HTTP/1.1 200 OK
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Allow: GET,OPTIONS,HEAD
Vary: Accept, Prefer
Content-Length: 924

{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "id": "http://example.org/annotations/?iris=0&page=0",
    "type": "AnnotationPage",
    "partOf": {
        "id": "http://example.org/annotations/?iris=0",
        "total": 42023,
        "modified": "2016-07-20T12:00:00Z",
    },
    "startIndex": 0,
    "next": "http://example.org/annotations/?iris=0&page=1",
    "items": [
        {
            "id": "http://example.org/annotations/anno1",
            "type": "Annotation",
            "body": "http://example.net/body1",
            "target": "http://example.com/page1"
        },
        {
            "id": "http://example.org/annotations/anno2",
            "type": "Annotation",
            "body": {
                "type": "TextualBody",
                "value": "I like this!"
            },
            "target": "http://example.com/book1"
        },
        ...
        {
            "id": "http://example.org/annotations/anno50",
            "type": "Annotation",
            "body": "http://example.org/texts/description1",
            "target": "http://example.com/images/image1"
        }
    ]
}
4.4 Discovery of Annotation Containers

As the IRI for Annotation Containers may be any IRI, and it is unlikely that every Web Server will support the functionality, it is important to be able to discover the availability of these services.

Any resource may link to an Annotation Container when Annotations on the resource should be created within the referenced Container. This link is carried in an HTTP Link header and the value of the rel parameter must be http://www.w3.org/ns/oa#annotationService.

For HTML representations of resources, the equivalent link tag in the header of the document may also be used.

For an example image resource, a GET request and response with a link to the above Annotation Container might look like:

Request:

```
EXAMPLE 14

GET /images/logo.jpg HTTP/1.1
Host: example.com
```

Response:

```
EXAMPLE 15

HTTP/1.1 200 OK
Content-Type: image/jpeg
Link: <http://example.org/annotations/>; rel="http://www.w3.org/ns/oa#annotationService"
Allow: GET
Content-Length: 76983

[...]```

5. Creation, Updating and Deletion of Annotations
5.1 Create a New Annotation

New Annotations are created via a POST request to an Annotation Container. The Annotation, serialized as JSON-LD, is sent in the body of the request. All of the known information about the Annotation should be sent, and if there are already IRIs associated with the resources, they should be included. The serialization should use the Web Annotation JSON-LD profile, and servers may reject other contexts even if they would otherwise produce the same model. The server may reject content that is not considered an Annotation according to the Web Annotation specification [annotation-model].

Upon receipt of an Annotation, the server may assign IRIs to any resource or blank node in the Annotation, and must assign an IRI to the Annotation resource in the id property, even if it already has one provided. The server should use HTTPS IRIs when those resources are able to be retrieved individually. The IRI for the Annotation must be the IRI of the Container with an additional component added to the end.

The server may add information to the Annotation. Possible additional information includes the agent that created it, the time of the Annotation's creation, or additional types and formats of the constituent resources.

If the Annotation contains a canonical property, then that reference must be maintained without change. If the Annotation has an IRI in the id property, then it should be copied to the via property, and the IRI assigned by the server at which the Annotation will be available must be put in the id field to replace it.

The server must respond with a 201 Created response if the creation is successful, and an appropriate error code otherwise. The response must have a Location header with the Annotation's new IRI.

Request:
EXAMPLE 16

POST /annotations/ HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Length: 202

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "type": "Annotation",
  "body": {
    "type": "TextualBody",
    "value": "I like this page!"
  },
  "target": "http://www.example.com/index.html"
}

Response:

EXAMPLE 17

HTTP/1.1 201 CREATED
Allow: PUT,GET,OPTIONS,HEAD,DELETE,PATCH
Location: http://example.org/annotations/anno1
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Length: 287
ETag: "_87e52ce126126"

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/annotations/anno1",
  "type": "Annotation",
  "created": "2015-01-31T12:03:45Z",
  "body": {
    "type": "TextualBody",
    "value": "I like this page!"
  },
  "target": "http://www.example.com/index.html"
}
5.2 Suggesting an IRI for an Annotation

The IRI path segment that is appended to the Container IRI for a resource may be suggested by the Annotation Client by using the Slug HTTP header on the request when the resource is created. The server should use this name, so long as it does not already identify an existing resource, but may ignore it and use an automatically assigned name.

Request:

```
POST /annotations/ HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Length: 202
Slug: "my_first_annotation"

{
    "@context": "http://www.w3.org/ns/anno.jsonld",
    "type": "Annotation",
    "body": {
        "type": "TextualBody",
        "value": "I like this page!"
    },
    "target": "http://www.example.com/index.html"
}
```

Response:
5.3 Update an Existing Annotation

Annotations can be updated by using a PUT request to replace the entire state of the Annotation. Annotation Servers should support this method. Servers may also support using a PATCH request to update only the aspects of the Annotation that have changed, but that functionality is not specified in this document.

Replacing the Annotation with a new state must be done with the PUT method, where the body of the request is the intended new state of the Annotation. The client should use the If-Match header with a value of the ETag it received from the server before the editing process began, to avoid collisions of multiple users modifying the same Annotation at the same time. This feature is not mandatory to support, as not every system will have multiple users with the potential to change a single Annotation, or use cases might dictate situations in which overwriting is the desired behavior.

Servers should reject update requests that modify the values of the canonical or via properties, if they have been already set, unless business logic allows the request to be trusted as authoritatively correctly a previous error.
If successful, the server must return a 200 OK status with the Annotation as the body according to the content-type requested. As with creation, the server must return the new state of the Annotation in the response.

Request:

```
PUT /annotations/anno1 HTTP/1.1
Host: example.org
Accept: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
Content-Length: 294
If-Match: "_87e52ce126126"

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/annotations/anno1",
  "type": "Annotation",
  "created": "2015-02-01T10:13:40Z",
  "body": {
    "type": "TextualBody",
    "value": "I REALLY like this page!"
  }
}
```

Response:
5.4 Delete an Existing Annotation

Clients must use the DELETE HTTP method to request that an Annotation be deleted by the server. Annotation Servers should support this method. Clients should send the ETag of the Annotation in the If-Match header to ensure that it is operating against the most recent version of the Annotation.

If the DELETE request is successfully processed, then the server must return a 204 status response. The IRIs of deleted Annotations should not be re-used for subsequent Annotations. The IRI of the deleted Annotation must be removed from the Annotation Container it was created in. There are no requirements made on the body of the response, and it may be empty.

Request:

```
EXAMPLE 21

HTTP/1.1 200 OK
Content-Type: application/ld+json; profile="http://www.w3.org/ns/anno.jsonld"
ETag: "_87e52ce234234"
Link: <http://www.w3.org/ns/ldp#Resource>; rel="type"
Allow: PUT, GET, OPTIONS, HEAD, DELETE, PATCH
Vary: Accept
Content-Length: 331

{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/annotations/anno1",
  "type": "Annotation",
  "created": "2015-02-01T10:13:40Z",
  "modified": "2015-02-02T20:43:19Z",
  "body": {
    "type": "TextualBody",
    "value": "I REALLY like this page!"
  },
  "target": "http://www.example.com/index.html"
}
```
6. Error Conditions

This section is non-normative.

There are inevitably situations where errors occur when retrieving or managing Annotations. The use of the HTTP status codes below provides a method for clients to understand the reason why a request has failed. Some of the situations that might occur, and the preferred HTTP status code are given below. This list is intended to be informative and explanatory, rather than imposing additional requirements beyond those already established by HTTP.

<table>
<thead>
<tr>
<th>Code</th>
<th>Example Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>The Annotation Client sent a request which the Annotation Server cannot process due to the request not following the appropriate specifications.</td>
</tr>
<tr>
<td>401</td>
<td>The Annotation Client is not authorized to perform the requested operation, such as creating or deleting an Annotation, as it did not supply authentication credentials.</td>
</tr>
<tr>
<td>403</td>
<td>The Annotation Client is not authorized to perform the requested operation, as the authentication credentials supplied did not meet the requirements of a particular access control policy for the Annotation or Annotation Container.</td>
</tr>
<tr>
<td>404</td>
<td>The Annotation or Annotation Container requested does not exist.</td>
</tr>
<tr>
<td>405</td>
<td>The requested HTTP method is not allowed for the resource, such as trying to POST to an Annotation Container page, or trying to PATCH an Annotation when that functionality is not supported.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>406</td>
<td>The requested format for the Annotation or Annotation Container's representation is not available, for example if a client requested RDF/XML and the server does not support that (optional) transformation.</td>
</tr>
<tr>
<td>409</td>
<td>The Annotation Client tried to set or change a value that the server does not allow Clients to modify, such as the containment list of an Annotation Container or server set modification timestamps.</td>
</tr>
<tr>
<td>410</td>
<td>The Annotation is known to have existed in the past and was deleted.</td>
</tr>
<tr>
<td>412</td>
<td>The Annotation Client supplied an If-Match header that did not match the ETag of the Annotation being modified.</td>
</tr>
<tr>
<td>415</td>
<td>The Annotation Client sent an entity-body that is not able to be processed by the Server, such as non-Annotation or in a context that is unrecognized.</td>
</tr>
</tbody>
</table>

7. Containers for Related Resources

Annotations may have related resources that are required for their correct interpretation and rendering, such as content resources used in or as the Body, CSS stylesheets that determine the rendering of the annotation, SVG documents describing a non-rectangular region of a resource, and so forth. If these resources do not already have IRIs, then they need to be made available somewhere so that they can be referred to.

Annotation Servers may support the management of related resources independently from the Annotations. If a server supports the management of these resources, it should do this with one or more separate Containers. Resources that are not Annotations should not be included in an Annotation Container, as Annotation Clients would not expect to find arbitrary content when dereferencing the IRIs. Containers for related resources may contain both RDF Sources and Non-RDF Sources. No restrictions are placed on the type or configuration of the Container beyond those of the Linked Data Platform [ldp].

Containers for related resources must support the same HTTP methods as described above for the Annotation Container, and must support identifying their type with a Link header. The constrainedBy link header on the response when dereferencing the Container should refer to a server specific set of constraints listing the types of content that are acceptable.

A. Candidate Recommendation Exit Criteria

This section is non-normative.
For this specification to be advanced to Proposed Recommendation, there must be at least two independent implementations of each feature described below. Each feature may be implemented by a different set of products, and there is no requirement that any single product implement every feature.

Features

For the purposes of evaluating exit criteria, the following operations are considered as features:

- HTTP GET of an Annotation
- HTTP GET of an Annotation Collection
- HTTP GET of an Annotation Collection Page, with embedded Annotations
- HTTP GET of an Annotation Collection Page, without embedded Annotations
- POST of an Annotation to an Annotation Collection
- POST of an Annotation to an Annotation Collection, with a Slug to suggest the IRI
- PUT of an Annotation to update an existing Annotation in an Annotation Collection
- DELETE of an Annotation

Each feature must be implemented according to the requirements given in the specification, regarding the HTTP headers, status codes, and entity body. Software that does not alter its behavior in the presence or lack of a given feature is not deemed to implement that feature for the purposes of exiting the Candidate recommendation phase.

B. Changes from Previous Versions

This section is non-normative.

B.1 Changes from the Proposed Recommendation of 2017-01-17

No significant changes.

B.2 Changes from the Candidate Recommendation of 2016-09-06

Editorial changes in this specification from the Candidate Recommendation of 2016-09-06 are:

- Clarified which header requirements are appropriate for HEAD/GET, rather than OPTIONS.
- Clarified interaction of multiple preferences in a single request.
- Removed incorrect Prefer from the Vary header from the pages example.
- Summarized requirements from the Web Annotation Data Model for Collections and Pages
- Clarified text regarding URI requirements for Containers.
- Clarified use of modified with respect to Containers and Pages.
- Clarified use of canonical with update operations.
- Fixed RFC 7230 / RFC 7232 typo.
- Clarified use of ETag and If-Match.

**B.3 Changes from the Candidate Recommendation of 2016-07-12**

Editorial changes in this specification from the Candidate Recommendation of 2016-07-12 are:

- Added CR Exit Criteria
- Editorial restructuring of the Pagination content.
- Clarified requirements for Annotation Server's representation preference handling.
- Explained the proper use of the Prefer header when the client has multiple preferences.

**B.4 Changes from the Working Draft of 2016-03-31**

Significant technical changes in this specification from the Working Draft Published of 2016-03-31 are:

- Use ActivityStreams based Paging mechanism to replace LDP Paging, to allow for in-page order.
- Recommend the use of HTTPS over HTTP.
- Rename PreferContainedURIs to PreferContainedIRIs.
- Add recommendation for Accept-Post, with example.
- Clarify expected status codes for successful interactions.
- Restructure Container Retrieval section and promote Container Representations and Annotation Pages sections.

**C. Acknowledgments**
The Web Annotation Working Group gratefully acknowledges the contributions of the Open Annotation Community Group. The output of the Community Group was fundamental to the current data model and protocol.

The following people have been instrumental in providing thoughts, feedback, reviews, content, criticism and input in the creation of this specification:


D. References

D.1 Normative references

[RFC2119]

[activitystreams-core]
Activity Streams 2.0. James Snell; Evan Prodromou. W3C. 15 December 2016. W3C Candidate Recommendation. URL: https://www.w3.org/TR/activitystreams-core/

[activitystreams-vocabulary]

[annotation-model]
Web Annotation Data Model. Robert Sanderson; Paolo Ciccarese; Benjamin Young. W3C. W3C Recommendation. URL: https://www.w3.org/TR/annotation-model/

[annotation-vocab]
Web Annotation Vocabulary. Robert Sanderson; Paolo Ciccarese; Benjamin Young. W3C. W3C Recommendation. URL: https://www.w3.org/TR/annotation-vocab/

[cors]
[ldp]
Linked Data Platform 1.0. Steve Speicher; John Arwe; Ashok Malhotra. W3C. 26 February 2015. W3C Recommendation. URL: https://www.w3.org/TR/ldp/

/rfc3987

/rfc5988

/rfc7231

/rfc7232