Introduction

The *Competency Index for Linked Data* (CI) is comprised of a set of topically arranged assertions of the knowledge, skills and habits of mind required for professional practice in the area of Linked Data. The structure of the CI is as follows:

---

**1. Topic cluster**

**1.1. Topic**

1.1.1. Competency: Tweet-length assertion of knowledge, skill, or habit of mind

1.1.1.1. Benchmark: Action demonstrating accomplishment in related competencies

---

This structure is implemented on the Learning Resources page of the LD4PE website (1). CI’s further enhancement is expected to accumulate openly crowd-sourced expertise in the development process under the guidance of the project’s CI Editorial Board.

The primary goal of the RDF-modeled CI is to provide a means for mapping learning resources’ descriptions to the competencies that the resources address in order to assist in finding, identifying, selecting, obtaining, and exploring resources appropriate to specific learning needs. In the display of the CI on the Explore Learning Resources page (1), individual nodes of competencies and benchmarks are followed by a number (e.g. “(4)”) indicating the number of learning resource descriptions that match to the competency/benchmark.

The CI model and representation is based on the Achievement Standards Network (ASN) *Competency Index for the Library Field* (2). Additional information on the LD4PE project and its development can be found on the LD4PE website under the Briefing Papers section (3)—see in particular “LD4PE Overview” item (4).

1) [http://explore.dublincore.net/explore-learning-resources-by-competency/](http://explore.dublincore.net/explore-learning-resources-by-competency/)

2) [http://asn.jesandco.org/resources/D2527589](http://asn.jesandco.org/resources/D2527589) (See the *Competency Index for the Library Field* in the U.S. ASN-D2L triplestore—click the expansion icon for a full view of the individual assertions; click on a specific assertion to see the assertion's metadata description on the right.)

3) [http://explore.dublincore.net/theory/briefing-papers/](http://explore.dublincore.net/theory/briefing-papers/)

4) [http://explore.dublincore.net/theory/briefing-papers/ld4peoverview/](http://explore.dublincore.net/theory/briefing-papers/ld4peoverview/)
Competency Index for Linked Data

<table>
<thead>
<tr>
<th>COMPETENCY INDEX FOR LINKED DATA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. FUNDAMENTALS OF RESOURCE DESCRIPTION FRAMEWORK</td>
<td>2</td>
</tr>
<tr>
<td>2. FUNDAMENTALS OF LINKED DATA</td>
<td>3</td>
</tr>
<tr>
<td>3. RDF VOCABULARIES AND APPLICATION PROFILES</td>
<td>4</td>
</tr>
<tr>
<td>4. CREATING AND TRANSFORMING RDF DATA</td>
<td>6</td>
</tr>
<tr>
<td>5. INTERACTING WITH RDF DATA</td>
<td>6</td>
</tr>
<tr>
<td>6. CREATING LINKED DATA APPLICATIONS</td>
<td>10</td>
</tr>
</tbody>
</table>

1. Fundamentals of Resource Description Framework

1.1. Identity in RDF

1.1.1. Knows that anything can be named with Uniform Resource Identifiers (URIs), such as agents, places, events, artifacts, and concepts.

1.1.2. Understands that a "real-world" thing may need to be named with a URI distinct from the URI for information about that thing.

1.1.3. Recognizes that URIs are "owned" by the owners of their respective Internet domains.

1.1.4. Knows that Uniform Resource Identifiers, or URIs (1994), include Uniform Resource Locators (URLs, which locate web pages) as well as location-independent identifiers for physical, conceptual, or web resources.

1.2. RDF data model

1.2.1. Knows the subject-predicate-object component structure of a triple.

1.2.2. Understands that URIs and literals denote things in the world ("resources") real, imagined, or conceptual.

1.2.3. Understands the difference between literals and non-literal resources.

1.2.4. Understands that resources are declared to be members (instances) of classes using the property rdf:type.

1.2.5. Understands the use of datatypes and language tags with literals.

1.2.6. Understands blank nodes and their uses.

1.2.7. Understands that QNames define shorthand prefixes for long URIs.

1.2.8. Understands a named graph as one of the collection of graphs comprising an RDF dataset, with a graph name unique in the context of that dataset.

1.2.9. Understands how a namespace, informally used in the RDF context for a namespace URI or RDF vocabulary, fundamentally differs from the namespace of data attributes and functions (methods) defined for an object-oriented class.

1.2.9.1. Uses prefixes for URIs in RDF specifications and data.

1.2.10. Articulates differences between the RDF abstract data model and the XML and relational models.

1.2.11. Understands the RDF abstract data model as a directed labeled graph.
1.2.12. Knows graphic conventions for depicting RDF-based models.
   
   1.2.12.1. *Can use graphing or modeling software to share those models with others.*

1.3. Related data models

1.3.1. Grasps essential differences between schemas for syntactic validation (e.g., XML) and for inferencing (RDF Schema).

1.3.2. Differentiates hierarchical document models (e.g., XML) and graph models (RDF).

1.3.3. Understands how an RDF class (named set of things) fundamentally differs from an object-oriented programming class, which defines a type of object bundling "state" (attributes with data values) and "behavior" (functions that operate on state).

1.4. RDF serialization

1.4.1. Distinguishes the RDF abstract data model and concrete serializations of RDF data.

   1.4.1.1. *Expresses data in serializations such as RDF/XML, N-Triples, Turtle, N3, Trig, JSON-LD, and RDFa.*

1.4.2. Understands RDF serializations as interchangeable encodings of a given set of triples (RDF graph).

   1.4.2.1. *Uses tools to convert RDF data between different serializations.*

2. Fundamentals of Linked Data

2.1. Web technology


2.1.2. Understands that Linked Data (2006) extended the notion of a web of documents (the Web) to a notion of a web of finer-grained data (the Linked Data cloud).

2.1.3. Knows HyperText Markup Language, or HTML (1991+), as a language for "marking up" the content and multimedia components of Web pages.


2.1.5. Knows Hypertext Transfer Protocol, or HTTP (1991+), as the basic technology for resolving hyperlinks and transferring data on the World Wide Web.

2.1.6. Knows Representational State Transfer, or REST (2000) as a software architectural style whereby browsers can exchange data with web servers, typically on the basis of well-known HTTP actions.

2.2. Linked data principles

2.2.1. Knows Tim Berners-Lee’s principles of Linked Data: use URIs to name things, use HTTP URIs that can be resolved to useful information, and create links to URIs of other things.
2.2.2. Knows the "five stars" of Open Data: put data on the Web, preferably in a structured and preferably non-proprietary format, using URIs to name things, and link to other data.

2.3. Linked Data policies and best practices

2.3.1. Knows the primary organizations related to Linked Data standardization.
   2.3.1.1. Participates in developing standards and best practice with relevant organizations such as W3C.

2.4. Non-RDF Linked Data

3. RDF vocabularies and application profiles

3.1. Finding RDF-based vocabularies

3.1.1. Knows portals and registries for finding RDF-based vocabularies.
   3.1.1.1. Finds properties and classes in the Linked Open Vocabularies (LOV) observatory and explores their versions and dependencies.

3.2. Designing RDF-based vocabularies

3.2.1. Uses RDF Schema to express semantic relationships within a vocabulary.
   3.2.1.1. Correctly uses sub-class relationships in support of inference.
   3.2.1.2. Correctly uses sub-property relationships in support of inference.

3.2.2. Knows the naming conventions for RDF properties and classes.

3.2.3. Reuses published properties and classes where available.

3.2.4. Coins namespace URIs, as needed, for any new properties and classes required.
   3.2.4.1. Drafts a policy for coining URIs for properties and classes.
   3.2.4.2. Chooses "hash"- or "slash"-based URI patterns based on requirements.

3.2.5. Knows Web Ontology Language, or OWL (2004), as a RDF vocabulary of properties and classes that extend support for expressive data modeling and automated inferencing (reasoning).

3.2.6. Knows that the word "ontology" is ambiguous, referring to any RDF vocabulary, but more typically a set of OWL classes and properties designed to support inferencing in a specific domain.

3.2.7. Knows Simple Knowledge Organization System, or SKOS (2009), an RDF vocabulary for expressing concepts that are labeled in natural languages, organized into informal hierarchies, and aggregated into concept schemes.

3.2.8. Knows SKOS eXtension for Labels, or SKOS-XL (2009), a small set of additional properties for describing and linking lexical labels as instances of the class Label.

3.2.9. Understands that in a formal sense, a SKOS concept is not an RDF class but an instance and, as such, is not formally associated with a set of instances ("class extension").
3.2.10. Understands that SKOS can express a flexibly associative structure of concepts without enabling the more rigid and automatic inferences typically specified in a class-based OWL ontology.

3.2.11. Understands that in contrast to OWL sub-class chains, hierarchies of SKOS concepts are designed not to form transitive chains automatically because this is not how humans think or organize information.

3.3. Maintaining RDF vocabularies

3.3.1. Understands policy options for persistence guarantees.
   3.3.1.1. Can draft a persistence policy.

3.4. Versioning RDF vocabularies

3.4.1. Knows technical options for the form, content, and granularity of versions.

3.4.2. Understands the trade-offs between publishing RDF vocabularies in periodic, numbered releases versus more continual or incremental approaches.
   3.4.2.1. Can express and justify a versioning policy.

3.5. Publishing RDF vocabularies

3.5.1. Understands that to be "dereferencable", a URI should be usable to retrieve a representation of the resource it identifies.
   3.5.1.1. Ensures that when dereferenced by a Web browser, a URI returns a representation of the resource in human-readable HTML.
   3.5.1.2. Ensures that when dereferenced by an RDF application, a URI returns representation of the resource in the requested RDF serialization syntax.

3.5.2. Understands the typical publication formats for RDF vocabularies and their relative advantages.

3.5.3. Understands the purpose of publishing RDF vocabularies in multiple alternative formats using content negotiation.

3.6. Mapping RDF vocabularies

3.6.1. Understands that the properties of hierarchical subsumption within an RDF vocabulary -- rdfs:subPropertyOf and rdfs:subClassOf -- can also be used to express mappings between vocabularies.

3.6.2. Understands that owl:equivalentProperty and owl:equivalentClass may be used when equivalencies between properties or between classes are exact.

3.6.3. Recognizes that owl:sameAs, while popular as a mapping property, has strong formal semantics that can entail unintended inferences.

3.7. RDF application profiles

3.7.1. Identifies real-world entities in an application domain as candidates for RDF classes.

3.7.2. Identifies resource attributes and relationships between domain entities as candidates for RDF properties.

3.7.3. Investigates how others have modeled the same or similar application domains.
3.7.3.1. Communicates a domain model with words and diagrams.
3.7.3.2. Participates in the social process of developing application profiles.

4. Creating and transforming RDF Data

4.1. Managing identifiers (URIs)
4.1.1. Understands that to be "persistent", a URI must have a stable, well-documented meaning and be plausibly intended to identify a given resource in perpetuity.
4.1.2. Understands trade-offs between "opaque" URIs and URIs using version numbers, server names, dates, application-specific file extensions, query strings or other obsoletable context.
4.1.3. Recognizes the desirability of a published namespace policy describing an institution's commitment to the persistence and semantic stability of important URLs.

4.2. Creating RDF data
4.2.1. Generates RDF data from non-RDF sources.
4.2.2. Knows methods for generating RDF data from tabular data in formats such as Comma-Separated Values (CSV).
4.2.3. Knows methods such as Direct Mapping of Relational Data to RDF (2012) for transforming data from the relational model (keys, values, rows, columns, tables) into RDF graphs.

4.3. Versioning RDF data

4.4. RDF data provenance

4.5. Cleaning and reconciling RDF data
4.5.1. Cleans a dataset by finding and correcting errors, removing duplicates and unwanted data.

4.6. Mapping and enriching RDF data
4.6.1. Uses available resources for named entity recognition, extraction, and reconciliation.

5. Interacting with RDF Data

5.1. Finding RDF Data
5.1.1. Knows relevant resources for discovering existing Linked Data datasets.
5.1.1.1. Retrieves and accesses RDF data from the "open Web".
5.1.2. Monitors and updates lists which report the status of SPARQL endpoints.
5.1.3. Uses available vocabularies for dataset description to support their discovery.
5.1.4. Registers datasets with relevant services for discovery.

5.2. Processing RDF data using programming languages
5.2.1. Understands how components of the RDF data model (datasets, graphs, statements, and various types of node) are expressed in the RDF library of a given programming language by constructs such as object-oriented classes.

5.2.1.1. Uses an RDF programming library to serialize RDF data in available syntaxes.

5.2.1.2. Uses RDF-specific programming methods to iterate over components of RDF data.

5.2.1.3. Uses RDF-library-specific convenience representations for common RDF vocabularies such as RDF, Dublin Core, and SKOS.

5.2.1.4. Programatically associates namespaces to prefixes for use in serializing RDF or when parsing SPARQL queries.

5.2.1.5. Uses RDF programming libraries to extract RDF data from CSV files, databases, or web pages.

5.2.1.6. Uses RDF programming libraries to persistently stores triples in memory, on disk, or to interact with triple stores.

5.2.1.7. Programatically infers triples using custom functions or methods.

5.2.2. Understands how the pattern matching of SPARQL queries can be expressed using functionally equivalent constructs in RDF programming libraries.

5.2.2.1. Uses RDF-specific programming methods to query RDF data and save the results for further processing.

5.2.2.2. Uses utilities and convenience functions the provide shortcuts for frequently used patterns, such as matching the multiple label properties used in real data.

5.2.2.3. Uses RDF libraries to process various types of SPARQL query result.

5.3. Querying RDF Data

5.3.1. Understands that a SPARQL query matches an RDF graph against a pattern of triples with fixed and variable values.

5.3.2. Understands the basic syntax of a SPARQL query.

5.3.2.1. Uses angle brackets for delimiting URIs.

5.3.2.2. Uses question marks for indicating variables.

5.3.2.3. Uses PREFIX for base URIs.

5.3.3. Demonstrates a working knowledge of the forms and uses of SPARQL result sets (SELECT, CONSTRUCT, DESCRIBE, and ASK).

5.3.3.1. Uses the SELECT clause to identify the variables to appear in a table of query results.

5.3.3.2. Uses the WHERE clause provide the graph pattern to match against the graph data.

5.3.3.3. Uses variables in SELECT and WHERE clauses to yield a table of results.

5.3.3.4. Uses ASK for a True/False test for a match to a query pattern.
5.3.3.5. Uses DESCRIBE to extract a single graph containing RDF data about resources.
5.3.3.6. Uses CONSTRUCT to extract and transform results into a single RDF graph specified by a graph template.
5.3.3.7. Uses FROM to formulate queries with URLs and local files.

5.3.4. Understands how to combine and filter graph patterns using operators such as UNION, OPTIONAL, FILTER, and MINUS.
5.3.4.1. Uses UNION to formulate queries with multiple possible graph patterns.
5.3.4.2. Uses OPTIONAL to formulate queries to return the values of optional variables when available.
5.3.4.3. Uses FILTER to formulate queries that eliminate solutions from a result set.
5.3.4.4. Uses NOT EXISTS to limit whether a given graph pattern exists in the data.
5.3.4.5. Uses MINUS to remove matches from a result based on the evaluation of two patterns.
5.3.4.6. Uses NOT IN to restrict a variable to not being in a given set of values.

5.3.5. Understands the major SPARQL result set modifiers, e.g., to limit or sort results, or to return distinct results only once.
5.3.5.1. Uses ORDER BY to define ordering conditions by variable, function call, or expression.
5.3.5.2. Uses DISTINCT to ensure solutions in the sequence are unique.
5.3.5.3. Uses OFFSET to control where the solutions processed start in the overall sequence of solutions.
5.3.5.4. Uses LIMIT to restrict the number of solutions processed for query results.
5.3.5.5. Uses projection to transforms a solution sequence into one involving only a subset of the variables.

5.3.6. Understands the use of SPARQL functions and operators.
5.3.6.1. Uses the regular expression (regex()) function for string matching.
5.3.6.2. Uses aggregates to apply expressions over groups of solutions (GROUP BY, COUNT, SUM, AVG, MIN) for partitioning results, evaluating projections, and filtering.
5.3.6.3. Uses the lang() function to return the language tag of an RDF literal.
5.3.6.4. Uses the langMatches() function to match a language tag against a language range.
5.3.6.5. Uses the xsd:decimal(expn) function to convert an expression to an integer.
5.3.6.6. **Uses the GROUP BY clause to transforms a result set so that only one row will appear for each unique set of grouping variables.**

5.3.6.7. **Uses the HAVING clause to apply a filter to the result set after grouping.**

5.3.7. Differentiates between a Default Graph and a Named Graph, and formulates queries using the GRAPH clause.

5.3.7.1. **Formulates advanced queries using FROM NAMED and GRAPH on local data.**

5.3.7.2. **Formulates advanced queries using FROM NAMED on remote data.**

5.3.7.3. **Formulates advanced queries on data containing blank nodes.**

5.3.7.4. **Formulates advanced queries using subqueries.**

5.3.8. Uses a temporary variable to extend a query.

5.3.9. Understands the role of Property Paths and how they are formed by combining predicates with regular expression-like operators.

5.3.10. Understands the concept of Federated Search.

5.3.10.1. **Formulates advanced queries on a remote SPARQL endpoint using the SERVICE directive.**

5.3.10.2. **Uses federated query to query over a local graph store and one or more other SPARQL endpoints.**

5.3.10.3. **Pulls data from a different SPARQL endpoints in one single query using the SERVICE directive.**

5.3.11. Converts/manipulates SPARQL query outputs (RDF-XML, JSON) to the exact format required by a third party tools and APIs.

5.3.12. Reads and understands high-level descriptions of the classes and properties of a dataset in order to write queries.

5.3.13. Uses available tools, servers, and endpoints to issue queries against a dataset.

5.3.13.1. **Execute SPARQL queries using the Jena ARQ command-line utility.**

5.3.13.2. **Queries multiple local data files using ARQ.**

5.3.13.3. **Uses ARQ to evaluate queries on local data.**

5.3.13.4. **Uses Fuseki server to evaluate queries on a dataset.**

5.3.13.5. **Queries multiple data files using Fuseki.**

5.3.13.6. **Accesses DBPedia's SNORQL/SPARQL endpoint and issues simple queries.**

5.4. **Visualizing RDF Data**

5.4.1. Uses publicly available tools to visualize data.

5.4.1.1. **Uses Google FusionTables to create maps and charts.**

5.4.2. Distills results taken from large datasets so that visualizations are human-friendly.

5.4.3. Converts/manipulates SPARQL query outputs (RDF-XML, JSON) to the exact format required by a third party tools and APIs.
5.5. Reasoning over RDF data

5.5.1. Understands the principles and practice of inferencing.
  5.5.1.1. Uses common entailment regimes and understands their uses.

5.5.2. Understands the role of formally declared domains and ranges for inferencing.

5.5.3. Understands how reasoning can be used for integrating diverse datasets.

5.5.4. Knows that Web Ontology Language (OWL) is available in multiple "flavors" that are variously optimized for expressivity, performant reasoning, or for applications involving databases or business rules.

5.5.5. Understands that OWL Full supports all available constructs and is most appropriately used when reasoning performance is not a concern.

5.6. Assessing RDF data quality

5.7. RDF Data analytics

5.7.1. Uses available ontology browsing tools to explore the ontologies used in a particular dataset.

5.8. Manipulating RDF Data

5.8.1. Knows the SPARQL 1.1 Update language for updating, creating, and removing RDF graphs in a Graph Store.
  5.8.1.1. Uses INSERT/DELETE to update triples.
  5.8.1.2. Uses a CONSTRUCT query to preview changes before executing an INSERT/DELETE operation.

5.8.2. Knows the SPARQL 1.1 Graph Store HTTP protocol for updating graphs on a Web server (in "RESTful" style).
  5.8.2.1. Uses GET to retrieve triples from a default graph or a named graph.
  5.8.2.2. Uses PUT to insert set of triples into a new graph (or replace an existing graph).
  5.8.2.3. Uses DELETE to remove a graph.
  5.8.2.4. Uses POST to add triples to an existing graph.
  5.8.2.5. Uses proper syntax to request specific media types, such as Turtle.

5.8.3. Understands the difference between SQL query language (which operates on database tables) and SPARQL (which operates on RDF graphs).

6. Creating Linked Data applications

6.1. Storing RDF data