YARRRML + LDES: Simultaneously Lowering Complexity from Knowledge Graph Generation and Publication

Gerald Haesendonck1*, Ben De Meester1, Julián Andrés Rojas Meléndez1, Dylan Van Assche1 and Pieter Colpaert1

1IDLab, Dep. of Electronics and Information Systems, Ghent University – imec, Zwijnaarde, Belgium

Abstract
Linked Data Event Streams (LDES) is an advanced Knowledge Graph (KG) publication specification aimed at continuous data source replication and synchronization with benefits such as data entities versioning and history retention while providing a self-descriptive API. However, building an LDES requires a high level of expertise in the Semantic Web ecosystem. In this demo paper, we show how we lower the complexity and need for expertise when using a more advanced KG publication method such as LDES by providing an extension point to YARRRML, a human-friendly way to configure KG generation via RML. Integrated in Matey, an online YARRRML editor, we show how little effort (adding five characters to the YARRRML syntax in the simplest case) allows (re)generating multiple versions of a KG as an Event Stream. As such, this extension provides an easy-to-use starting point for anyone wanting to create an LDES from non-semantic data.

Keywords
Knowledge Graph, YARRRML, LDES

1. Introduction
Linked Data Event Streams (LDES)1 [1] models publishing changes in Knowledge Graph (KG) data entities structured as an append-only log, allowing clients to replicate the history of a data set and efficiently synchronize with its latest changes. LDES has proven to be useful for integrating and publishing data in different domains such as base registries [1], cultural heritage collections [2], and marine georeferenced locations, boundaries, and regions for scientific and educational purposes [3]. LDES is promoted in a European data interoperability context2 and the government of Flanders started publishing data as LDES3
However, building an LDES requires a high level of expertise: (i) it requires advanced knowledge of Semantic Web concepts and technologies, and (ii) existing software for publishing LDESe\textsuperscript{4} \textsuperscript{5} start from version objects already formatted as LDES members, thus requiring an extra step to perform before data can be ingested. This inhibits uptake of the LDES specification and constitutes a problem for advanced KG publication methods in general.

In this demo paper, we show how we lower the complexity and need for expertise when using a more advanced KG publication method such as LDES by providing an intuitive way to configure LDES generation. For this, we rely on the RDF Mapping Language (RML) \textsuperscript{4} as a commonly used declarative way to describe a KG construction process. Namely, we extended YARRRML \textsuperscript{5}, a human-friendly representation for RML. We chose YARRRML as it has multiple extensions already \textsuperscript{6, 7} and is used in knowledge graph cloud services such as Google’s Enterprise Knowledge Graph\textsuperscript{6}.

Where others extend YARRRML to stay in sync with the underlying mapping language features, our extension adds no additional features but instead provides an easy-to-use starting point for anyone wanting to create an LDES from existing data.

Integrated in Matey, an online YARRRML editor, we show how little effort (adding five characters to the YARRRML syntax in the simplest case) allows (re)generating multiple versions of KG data entities as an Event Stream.

2. Adding an LDES Extension Point to YARRRML

Our YARRRML extension generates all needed RML rules that comply to an existing method for generating LDES using RML \textsuperscript{8}. A full specification and examples are available online at https://rml.io/yarrrml/spec/ldes/. We introduce a single extension point: adding the ldes key to the subjects collection in YARRRML is enough to define RML mappings generating a valid minimalist LDES with reasonable defaults. Subkeys allow further configuration to customize the resulting LDES.

We include the following properties of an ldes:EventStream instance according to the LDES specification\textsuperscript{7} tree:shape, ldes:timestampPath, and ldes:versionOfPath. In YARRRML they can be configured using the subkeys shape, timestampPath and versionOfPath respectively. The IRI to identify the ldes:EventStream instance can be configured with the id sub-key. We do not support LDES fragmentation, pagination and retention policies because they are typically use case specific and can easily be obtained by using existing LDES servers or clients once a basic LDES is available.

Finally, we add the watchedProperties sub-key. This key describes which fields in the original data to watch, as only new values in all given fields trigger the generation of a new tree:member in the LDES.

The generated RML rules use a function to take the watchedProperties values into account when generating a new tree:member, as introduced by Van Assche et al \textsuperscript{8}. For example,
<table>
<thead>
<tr>
<th>SensorID</th>
<th>Timestamp</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2023-01-01T08:00:00</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>2023-01-01T08:00:00</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2023-01-01T09:00:00</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2023-01-01T09:00:00</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 1
Example data: sensors and their temperature readings

```
subjects:
  - value: ex:$SensorID

targets:
  - [ out.ttl-void, turtle ]

ldes:
  id: ex:myldes

watchedProperties: [$(SensorID), $(Timestamp), $(Temperature)]

shape: ex:shape.shacl

timestampPath: [ex:ts, $(Timestamp), xsd:dateTime]

versionOfPath: [ex:hasVersion, ex:$SensorID]
```

Listing 1: Example of a subjects section with the ldes key and properties integrated.

watchedProperties: [$(Temperature)] results in a new LDES member only when the value of field Temperature in the original data was not recorded before. Listing 1, lines 5 – 10, illustrates how we can generate an LDES from data shown in Table 1.

Compared to using RML directly to generate an LDES, our approach reduces technical overhead, e.g., adding an LDES configuration to YARRRML requires maximum 7 additional lines of code, but results in more than 40 additional triples in the RML mapping document\(^8\). By using ldes as key and naming sub-keys close to the corresponding LDES properties, we provide an intuitive and simple way to configure LDES generation. We implemented the LDES extension using the YARRRML Parser\(^9\) which translates YARRRML rules into RML.

### 3. Matey: Human-friendly KG Generation & Publication

To demonstrate our extension, we extended Matey\(^10\), an open-source browser-based application that helps one write YARRRML rules, publicly available at https://rml.io/yarrrml/matey/.

We added a basic LDES example (loadable by clicking the button [Basic LDES]) which loads a CSV file with temperature readings and YARRRML mappings, and generates an LDES based on temperature changes when clicking the [Generate LD] button (Figure 1). Because we only want to generate members when the temperature changes, there is no member for sensor 2’s

---

\(^8\)Examples of YARRRML rules and their corresponding RML rules generating LDES can be found at https://github.com/RMLio/yarrrml-ldes-extension/tree/main/ldes, all folders contain links to YARRRML files (.yaml) and their corresponding RML files (.rml.ttl).

\(^9\)https://github.com/RMLio/yarrrml-parser/releases/tag/v1.5.2

\(^10\)https://rml.io/yarrrml/matey/
Figure 1: This Matey demo demonstrates how generating KG versions using the \texttt{ltes} key within YARRRML introduces very little additional complexity.

last reading. Clicking [Generate RML] shows the corresponding RML mapping rules which are applied on the data. When clicking [Generate LD] again, no members are generated at all, since all members are already generated. This demonstrates the feature of adding members to a previously generated LDES without violating the \texttt{watchedProperties} constraints. Clicking [Reset state] clears the state, after which the LDES can be re-generated.

4. Conclusion

By extending YARRRML to configure how to generate an LDES from existing data we lower the technical complexity of a task that requires domain-specific knowledge. Future research is needed to go beyond basic LDES generation and explore more advanced features such as fragmentation. We expect this type of work to (i) increase LDES uptake and (ii) inspire other (YARRRML-like) extensions to lower complexity for any KG generation or publication method.

Acknowledgments The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10), the Flemish Smart Data Space
References


