



Collaborative Project

GeoKnow - Making the Web an Exploratory for Geospatial Knowledge

Project Number: 318159

Start Date of Project: 2012/12/01

Duration: 36 months

Deliverable 4.2.1 Spatial authoring widget set

Dissemination Level	Public
Due Date of Deliverable	Month 20, 31/07/2014
Actual Submission Date	Month 24, 30/11/2014
Work Package	WP4, Spatial-Semantic Browsing, Visualisation and Authoring Interfaces
Task	T4.2
Type	Prototype Release
Approval Status	Approved
Version	1.0
Number of Pages	20
Filename	D4.2.1_Spatial_authoring_widget_Set.pdf

Abstract: In this deliverable we present a novel approach to Web-based RDF authoring. We show how our system makes it possible to easily integrate our own and third-party widgets, and we demonstrate the system in a number of use cases.

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/her sole risk and liability.



Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)

History

Version	Date	Reason	Revised by
0.0	03/03/2014	Initial Version	Jens Lehmann
0.1	10/03/2014	Draft of Related Work	Clemens Hoffmann
0.2	15/05/2014	Draft of Widget System Descriptions	Clemens Hoffmann
0.3	20/06/2014	Draft of REX annotation system	Claus Stadler
0.4	08/08/2014	Added use case Examples for REX	Claus Stadler
0.5	12/11/2014	Revised Widget System Description	Clemens Hoffmann
0.6	03/12/2014	Finalization	Claus Stadler
0.7	05/12/2014	Proof reading	Jens Lehmann
0.8	08/12/2014	Peer review	Jon Jay Le Grange
0.9	12/12/2014	Addressed peer review comments	Claus Stadler
1.0	16/12/2014	Approval	Jens Lehmann

Author List

Organization	Name	Contact Information
InfAI	Claus Stadler	cstadler@informatik.uni-leipzig.de
InfAI	Clemens Hoffmann	cannelony@gmail.com
InfAI	Jens Lehmann	lehmann@informatik.uni-leipzig.de

Executive Summary

This deliverable describes the RDF Edit eXtension (REX) system, which provides a flexible approach to form based RDF editing by means of a set of HTML attributes that are implemented as directives using the AngularJS framework. Web development concepts and frameworks have matured significantly in recent years, and with modern frameworks, such as AngularJS, a vast amount of widgets can be readily re-used as if they were a native part of HTML. The most significant contribution of this work is tackling the challenge of how re-use of widgets built on such state-of-the-art technologies can be facilitated in RDF editing scenarios. In addition, we created two widgets, one for editing geometric data on a map, and another one for editing RDF terms. With this work, we also cover how REX-annotated forms can interact with SPARQL endpoints for prefilling out fields and writing back the modifications. As part of these efforts, the Jassa library, developed in D4.1.1, was extended with features related to prefix processing, Talis RDF JSON (de-)serialization and SPARQL1.1/Update.

Abbreviations and Acronyms

LOD	Linked Open Data
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
JSON	JavaScript Object Notation
Jassa	Javascript Suite for Sparql Access
REX	RDF Edit eXtensions

Table of Contents

1	Introduction	5
1.1	Goals	6
1.2	Installation and Demonstration	6
2	RDF Edit eXtensions	7
2.1	Core HTML Attributes	7
2.2	Accessing the data and corresponding modifications	8
2.3	Conditional RDF generation	9
2.4	Data Lookups for prefilling forms	9
2.5	Renaming resources	10
2.6	Synchronization of models and converting values	10
3	Widgets	12
3.1	RDF Term Input	12
3.2	Geometry Input	12
4	Related Work	14
4.1	RDForm	14
4.2	RDFauthor	14
4.3	Backbone Forms and Vie.js	15
5	Conclusions and Future Work	16
6	Licence	17
	Appendices	18
A	Rex Directive Reference Table	18
B	Synchronization Directives Reference Table	20

Person data

By default the URI is generated from given name and surname, unless overridden by the user.

URI:

Given Name:

Surname:

E-Mail:

Gender:

Birthdate:

May 1984

Mon	Tue	Wed	Thu	Fri	Sat	Sun
18	30	01	02	03	04	06

Output

```

1 dbr:ClausStadler
2   dbo:birthDate "1984-05-
3   11T22:00:00.000Z"^^<xsd:date> ;
4   rdf:type <foaf:Person> ;
5   foaf:gender <http://male> ;
6   foaf:givenName "Claus" ;
7   foaf:mbox "cstadler@informatik.uni-
8   leipzig.de" ;
9   foaf:surname "Stadler" ;

```

[Perform Update](#)

Figure 1: An example form for capturing basic information about a person

1 Introduction

RDF is designed to provide a uniform data model for representing knowledge across domain boundaries. However, sustaining this flexibility in the context of RDF editing systems has turned out to be a difficult task. While there exist *applications* for editing RDF data, such as Protege¹, there is a lack of *components* for the light-weight integration of such capabilities into Web applications. In this document, we present a novel approach to RDF authoring, which, to the best of our knowledge, delivers unprecedented flexibility and ease in converting form data to and from RDF and demonstrate its applicability in various Use Cases. Our solution is based on using the AngularJS framework² to define a custom set of HTML attributes which are used to give values of form controls in the DOM-tree an RDF context. While this approach is similar to RDFa³, there are two important distinctions: First, RDFa is more suited for static HTML documents whereas our annotations are processed by the AngularJS framework at runtime based on the state of the DOM tree. Second, RDFa is aligned with the HTML standard, whereas our approach exploits concepts of AngularJS and builds upon its abstractions, most prominently the one introduced by *ngModel*. Our framework makes it easy to (a) reflect pending changes to resources immediately by keeping dependent views in sync as well as (b) process changes with SPARQL 1.1 Update⁴ requests. This allows, for example, a map widget to update itself immediately when the user edits the geo-coordinate of a shown resource. Note, that we use the terms widget and (form) control interchangeably. The structure of this prototype deliverable is as follows: In the remainder of this section, we first summarize the goals and achievements of this deliverable, followed by a brief summary of how our project can be re-used in other projects. In [section 2](#) we introduce concepts essential for the RDF authoring domain. Our widgets created for RDF editing are presented in [section 3](#). In [section 4](#) we review current approaches to RDF authoring and discuss their strengths and weaknesses. Finally, in [section 5](#) we conclude the deliverable.

¹<http://protege.stanford.edu/>

²<https://angularjs.org/>

³<http://www.w3.org/TR/rdfa-syntax/>

⁴<http://www.w3.org/TR/sparql11-update/>

1.1 Goals

To illustrate the goal of this deliverable, [Figure 1](#) juxtaposes a simple HTML form with its state converted to RDF. The main challenge tackled by this work is to devise a simple yet generic solution for overcoming this gap. In more detail, the goals were as follows:

- Development of an AngularJS-based annotation system that enables two-way data binding of the state of an HTML form with the one of an RDF graph. This means, if either changes, the other must be updated.
- Support for prefilling out a form based on an RDF resource's state in regard to a given data source, such as a SPARQL endpoint or Linked Data URL.
- Support for the integration of third party AngularJS widgets: AngularJS wrappers exist for a wide range of "legacy" components (such as jQuery and jQuery UI), and in many cases wrappers can be created with little effort.
- SPARQL 1.1 Update support to persist changes to an underlying RDF store.
- Support for RDF prefixes/namespaces to increase convenience and for possibly broader acceptance by the community.
- Reuse of the *Javascript Suite for Sparql Access* (Jassa) library, developed for D4.1.1, and the addition of needed functionality.

1.2 Installation and Demonstration

Our project is available for developers as a bower artifact, and can thus be used in a *bower*, *npm* and *grunt/gulp* environment⁵. The statement shown below fetches the dependency.

```
1 bower install jassa-ui-angular-edit
```

Consequently, references from a `bower.json` file work as follows. Make sure to adjust all present versions as needed.

```
1 {
2   "name": "my-application",
3   "version": "0.0.1",
4   "dependencies": {
5     "jassa-ui-angular-edit": "1.0.0",
6   }
7 }
```

The source code of the project is available on GitHub⁶, whereas the bower releases are currently performed in a separate repository⁷. The source code of demos are available in the `demo` folder and hosted versions can be found on the GeoKnow website⁸.

⁵For details, see <http://bower.io/>, <https://www.npmjs.com/>, <http://gruntjs.com/> and <http://gulpjs.com/>

⁶<https://github.com/GeoKnow/Jassa-UI-Angular/tree/master/jassa-ui-angular-edit>

⁷<https://github.com/GeoKnow/Jassa-UI-Angular-Edit-Bower>

⁸<http://js.geoknow.eu/demos/edit>

2 RDF Edit eXtensions

In this section we explain our annotation system, which we refer to as *RDF edit extensions* (REX) as we are extending HTML forms with support for RDF editing. AngularJS makes it possible to annotate HTML form controls with `ng-model="var"`, which declares a JavaScript variable `var` as the model of the control. When the user enters input, the model's value is changed accordingly, and when the model changes, the form control updates itself to reflect the new state. Our approach is based on the introduction of a set of annotations which give model values an RDF context. In the subsequent sections we explain the REX attributes.

2.1 Core HTML Attributes

Consider the example in [Listing 1](#).

```
1 <div rex-context
2   rex-prefix="'dbr: http://dbpedia.org/resource/'" rex-subject="'dbr:Chemnitz'"
3   <input
4     type="text" ng-model="model"
5     rex-property="'rdfs:label'"
6     rex-object rex-type="uri" rex-value="label">
7 </div>
```

Listing 1: A minimal REX example

First of all, the `rex-context` is used to activate the REX system on a DOM element and its children. The presence of the attribute creates the `rexContext` object in the corresponding scope, whose purpose is to keep track of (a) RDF values referenced in form controls, (b) initial data for prefilling forms, (c) modifications to the initial data for computing diffs, and (d) the final RDF data.

Within an `rex-context` section, the basic annotations are: `rex-subject="subjectIri"`, `rex-predicate="predicateIri"`, and `rex-object="objectIndex"`. While `rex-subject` and `rex-predicate` accept strings for IRIs as values, `rex-object` takes an *index*. The index refers to the *i*th RDF term for a given subject and predicate. RDF terms are represented as Talis RDF JSON⁹ objects that comprise four *components* of type string, namely `rex-type`, `rex-value`, `rex-lang`, and `rex-datatype`. In addition, `rex-deleted` can be used to flag an object and thus the corresponding triple as deleted. We refer to the combination of a subject IRI, predicate IRI, object index and component name as a *coordinate*. A coordinate uniquely references a primitive value in an RDF graph. The main reasons why we chose Talis RDF JSON as an *intermediate* representation instead of JSON-LD¹⁰ are: The simple nested structure of Talis RDF JSON is exactly the structure that is being conveyed by REX. Conversely, this structure is easy to use with Angular `ng-repeat` loops in combination with REX annotations. Note, that the *final* RDF graph that corresponds to the form state can be serialized in any RDF syntax, including JSON-LD.

For convenience, prefixes can be used to abbreviate IRIs. By default, all prefixes of the RDFa initial context¹¹ are readily available. Custom namespaces can be enabled by specifying `rex-prefix="prefixStr"`, where the argument must follow the same syntax as in RDFa¹². Also, REX features the short hands `rex-iri="model"` and `rex-literal="model"`. The first one becomes expanded to `rex-type="uri"`, `rex-value="model"`, whereas the latter becomes `rex-type="literal"` and `rex-value="model"`. The

⁹<http://www.w3.org/TR/rdf-json/>

¹⁰<http://json-ld.org/>

¹¹<http://www.w3.org/2011/rdfa-context/rdfa-1.1>

¹²<http://www.w3.org/TR/rdfa-syntax/#A-prefix>

final short hand is `rex-typeof="iriModel"` which becomes expanded to `rex-predicate="'rdf:type'", rex-iri="iriModel"`.

Note, that when multiple attributes occur on the same element, the order of processing depends on their priority value. A reference sheet is available in [Appendix A](#).

2.2 Accessing the data and corresponding modifications

The `rexContext` object is a container for all relevant information about the state of the form and performed modifications. It provides the following attributes:

- `.base` The base RDF graph, in Talis RDF JSON, which holds information about the resources referenced by the form controls. This information is automatically updated accordingly when the set of subject resources and or lookup functions change.
- `.override` A (partial) Talis RDF JSON object which holds the state of the data entered into the form and which can be seen to “override” any base data.
- `.graph` The RDF graph that holds the *effective* set of triples. This is thus obtained by applying the overrides to the base data for all data referenced by coordinates of the form.
- `.diff.added` An array containing the triples that were added in regard to the source data and referenced coordinates.
- `.diff.removed` An array containing the triples that were removed in regard to the source data and the referenced coordinates.

[Listing 2](#) shows how turtle representations and SPARQL Insert/Delete queries can be created from it.

```
1 <script type="text/javascript">
2   $scope.createInsertRequest = function(graph, prefixMapping) {
3     return graph
4       ? (new jassa.sparql.UpdateDataInsert(
5           jassa.sparql.QuadUtils.triplesToQuads(graph))
6           .asString(prefixMapping)
7           : '');
8   }; // For the delete version, just replace 'Insert' with 'Delete'
9
10  $scope.graphToTurtle = function(graph, prefixMapping) {
11    var talis = graph ?
12      jassa.io.TalisRdfJsonUtils.triplesToTalisRdfJson(graph)
13      : null;
14    return talis
15      ? jassa.io.TalisRdfJsonUtils.talisRdfJsonToTurtle(talis, prefixMapping)
16      : '';
17  }
18 </script>
19 <pre>{{graphToTurtle(rexContext.graph, rexPrefixMapping)}}</pre>
20 <pre>{{createInsertRequest(rexContext.diff.added, rexPrefixMapping)}}</pre>
21 <pre>{{createDeleteRequest(rexContext.diff.removed, rexPrefixMapping)}}</pre>
```

Listing 2: Creating triples from the source data

WKT <-> LonLat sync

Conditional generation of RDF using hidden fields

Output

```

1 dbr:Leipzig
2   rdf:type <dbo:Place> ;
3   geo:geometry "POINT(12.3833 51.3333)" ;
4   geo:lat "51.3333"^^<xsd:float> ;
5   geo:long "12.3833"^^<xsd:float> ;
6   .
7
```

Figure 2: A form for a geometry and the corresponding conditionally generated RDF

2.3 Conditional RDF generation

There are cases when RDF output should be constructed conditionally based on the form and validation state. For example, consider a form that allows a user to enter a WKT string, and corresponding WGS84 lat/long triples should be generated for valid POINT geometries, as depicted in [Figure 2](#). This can be accomplished using a combination of `ng-if` and hidden fields¹³, as shown in [Listing 3](#).

```

1 <script type="text/javascript">$scope.PointUtils = jassa.geo.PointUtils;</script>
2
3 <div rex-context>
4   <input type="text" ng-model="wkt" rex-predicate="'geo:geometry'" rex-literal>
5
6   <input ng-if="PointUtils.isWktPoint(wkt)" type="hidden"
7     rex-predicate="'geo:long'"
8     rex-object-literal="'' + PointUtils.wktToXy(wkt).x"
9     rex-datatype="'xsd:float'">
10
11   <input ng-if="PointUtils.isWktPoint(wkt)" type="hidden"
12     rex-predicate="'geo:lat'"
13     rex-object-literal="'' + PointUtils.wktToXy(wkt).y"
14     rex-datatype="'http://www.w3.org/2001/XMLSchema#float'"
15   >
16 </div>

```

Listing 3: Generating lat/long triples only for valid WKT strings

2.4 Data Lookups for prefilling forms

Forms can be connected to a data source using `rex-lookup="fn"`, where *fn* is a function that takes an IRI string as argument and yields a promise for an RDF graph object which implement the Graph interface¹⁴.

¹³The usage of hidden fields in this scenario seems idiomatic, although the approach would work with any HTML element, such as DIV and SPAN.

¹⁴<http://www.w3.org/TR/rdf-interfaces/>

Whenever the value of a `rex-subject` changes and a `rex-lookup` attribute is present, a lookup will be performed, and the data is converted to Talis RDF JSON and stored at `rexContext.json`. Listing 4 shows an example set up using a utility method from the Jassa library.

```
1 <script type="text/javascript">
2 var sparqlService = jassa.service.SparqlServiceBuilder
3   .http('http://dbpedia.org/sparql', ['http://dbpedia.org']).create();
4
5 $scope.lookupFn = function(node) {
6   return jassa.service.ServiceUtils.execDescribeViaSelect(sparqlService, [node]);
7 }
8 </script>
9
10 <div rex-context rex-lookup="lookupFn" rex-subject="'http://dbpedia.org/resource/
11   Leipzig'">
12   <!-- Within this block, models used as arguments for
13     rex-type, rex-value, rex-lang and rex-datatype will become initialized
14     with appropriate data from the lookup function. -->
15 </div>
```

Listing 4: Registering a lookup function for prefilling out forms based on subject resources

2.5 Renaming resources

While it may seem intriguing to make use of the `rex-subject` annotation for the purpose of renaming resources, this is not directly in the scope of the REX system. REX uses the concept of coordinates to give model values an RDF context. Under this perspective, coordinates are only *references* used to get/put model values from/into an RDF space, i.e. a component of a triple in an RDF graph. But coordinates are distinct from *operations* to be performed on an RDF graph.

Following this idea, renaming of resources has to be done using a custom function which is not part of the REX system, although the attributes of REX could be reused as shown in Listing 5.

```
1 <div rex-subject="oldIRI">
2   Enter a new IRI: <input type="text" ng-model="newIRI">
3   <button ng-click="doRename(rexSubject, newIRI)">
4 </div>
```

Listing 5: Example for renaming a resource

2.6 Synchronization of models and converting values

In the prior sections, we only discussed cases where a model value is placed directly into a corresponding Talis RDF JSON object. In this section, we show how to deal with cases where model values need to be converted. A simple use case is to combine the date picker of AngularUI Bootstrap¹⁵ with REX, as shown in Listing 6. Applying the date picker to an input element causes the model values to become JavaScript object of type *Date*. These values need to be converted to ISO8601 strings in RDF serializations. Because JavaScript does not

¹⁵<http://angular-ui.github.io/bootstrap/>

provide native functionality for this conversion, the *moment.js library*¹⁶ is used. Our approach is to introduce a set of **sync-*** annotations for synchronizing model values: The source and target of a sync are declared using **sync-source**, **sync-target**. With **sync-to-target** and/or **sync-to-source** any changes in the values are propagated in the respective direction. Both attributes take a conversion function as an optional argument. If desired, syncing in one of the directions can be enabled/disabled by specifying a condition expression using **sync-to-target-cond** and/or **sync-to-source-cond**. It is also possible to use a template string instead of a model as the source, which must be indicated by additionally using **sync-source-interpolate**.

For convenience, **rex-template="templateStr"** is available to copy values of the given template string into a model present on the same element. Internally, this attribute is expanded to **sync-source="templateStr"**, **sync-source-interpolate**, **sync-target="valueOfNgModelAttribute"**, and **sync-to-target**.

```
1 <script type="text/javascript">
2   $scope.dateToString = function(x) { return !x ? null : moment(x).toISOString(); };
3   $scope.parseDate = function(x) { return !x ? null : moment(x).toDate(); };
4 </script>
5
6 <div rex-predicate="'dbo:birthDate'"
7   rex-literal="birthDateStr" rex-datatype="'xsd:date'">
8   <input type="text" ng-model="birthDate"
9     sync-source="birthDate" sync-target="birthDateStr"
10    sync-to-target="dateToString" sync-to-source="parseDate"
11    datepicker-popup="dd-MMMM-yyyy" datepicker-options="dateOptions">
12 </div>
```

Listing 6: Two way binding of different models using conversions

¹⁶<http://momentjs.com/>

3 Widgets

In this section we briefly describe two widgets which we developed for REX use cases. Note, that these widgets can be used stand-alone without the need for any REX annotations.

3.1 RDF Term Input

The `rdf-term-input` widget, shown in [Figure 3](#) and [Listing 7](#), features generic editing of an RDF term, and is comprised of a term type selector and an input area. The term type can be set to *IRI*, *plain*, and *typed*. If *plain* is chosen, a combo-box for choosing an optional language tag is shown. In case of the *typed* option, a comb-box for choosing a mandatory datatype is displayed. The model uses JavaScript objects following the Talis RDF JSON format.

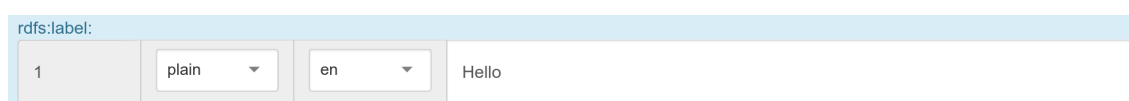


Figure 3: A generic widget for editing RDF terms

```

1 <script type="text/javascript">
2   $scope.model = {type: 'literal', value: 'Hello', lang: 'en' }
3 </script>
4 <rdf-term-input ng-model="model">

```

Listing 7: Usage of the RDF term input widget

3.2 Geometry Input

For geospatial use cases, we created the `geometry-input` directive based on OpenLayers¹⁷, depicted in [Figure 4](#). The widget displays a map view and supports several options for editing geometries, such as points and (non-)regular and polygons. The `ng-model` works with Well Known Text (WKT) string representations of the geometry. Usage of the directive is shown in [Listing 8](#).

```

1 <script type="text/javascript">
2   $scope.model = 'POINT (0 0)';
3 </script>
4 <geometry-input ng-model="model">

```

Listing 8: The geometry-input widget

¹⁷<http://openlayers.org/>

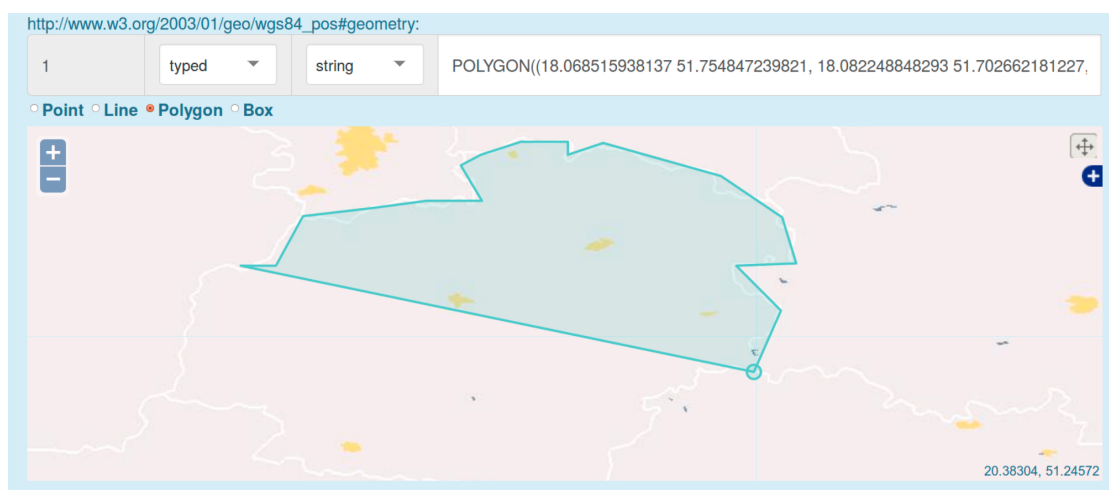


Figure 4: A generic widget for editing RDF terms

4 Related Work

In this section we review selected systems and tools that are related to our approach.

4.1 RForm

RForm¹⁸, depicted in Figure 5, is “a jQuery plugin for creating and editing RDF data in a clean and modern HTML form”. Technically, RForm is capable of compiling form templates, written in HTML with RDFa annotations, to DOM elements and attaching them to appropriate locations in a Web page using jQuery, as sketched in Listing 9 and Listing 10. JSON-LD is used for inserting existing data (prefilling the form) and for the output. Although the annotations of RForm are similar to those of REX, the main difference is, that RForm is neither based on AngularJS, nor does it mimic its model concept. For this reason, REX offers a significantly higher flexibility in creating custom data flows.

```
1 <form prefix="foaf http://xmlns.com/foaf/0.1/ rdfs http://www.w3.org/2000/01/rdf-  
  schema#">  
2   <legend>A person</legend>  
3   <div typeof="foaf:Person" resource="Person-{rdfs:label}">  
4     <label>The label</label>  
5     <input name="rdfs:label" datatype="xsd:string" />  
6   </div>  
7 </form>
```

Listing 9: An example form specification using RForm

```
1 $(document).ready(function(){  
2   $(".rdform").RForm({  
3     template: "templates/form.html",  
4  
5     submit: function() {  
6       console.log( JSON.stringify(this, null, '\t') );  
7     }  
8   });  
9 });
```

Listing 10: Integration of an RForm form into a Web page

4.2 RDFauthor

RDFauthor¹⁹ is a JavaScript library that adds a feature-rich authoring component to either a specified region of a Web page or its own window-like overlay, as illustrated in Figure 6.

RDFauthor itself ships with a set of widgets, such as a rich text editor, and a point geometry widget with geocoding support, and chooses the appropriate ones based on the current data being edited. Although it is possible to extend RDFauthor’s widget library with custom ones, changing the form layouts or injecting custom HTML annotations would require programmatic changes. For this reason, RDFauthor is not suited for creating custom forms.

¹⁸<https://github.com/simeonackermann/RForm>

¹⁹<https://github.com/AKSW/RDFauthor>

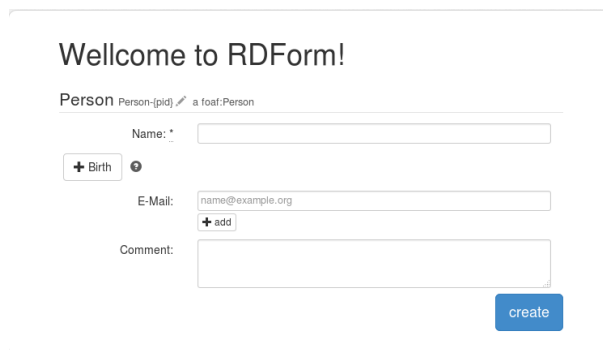


Figure 5: Screenshot of RForm (taken from its GitHub repo)

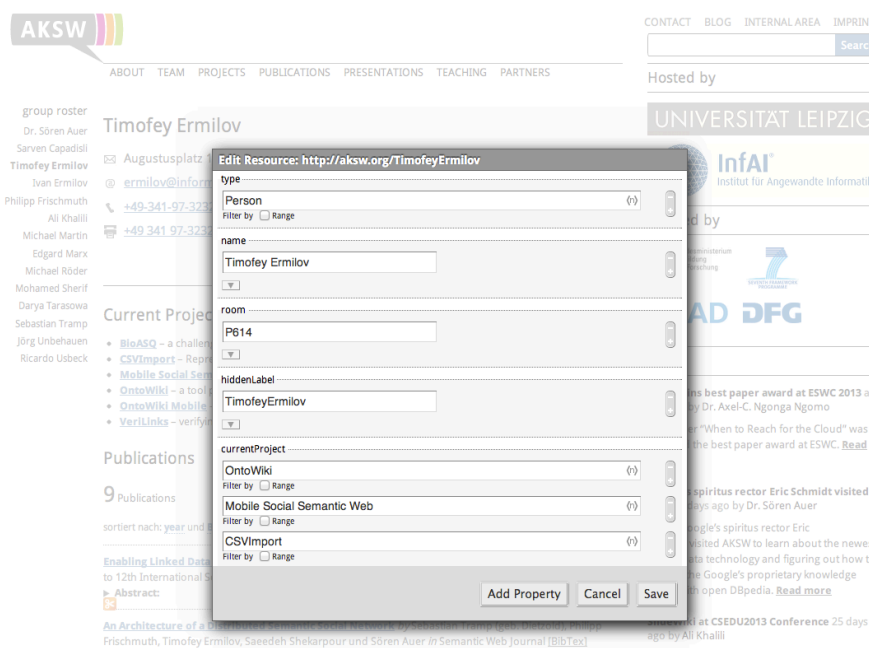


Figure 6: Screenshot of RDFAuthor

4.3 Backbone Forms and Vie.js

Backbone Forms²⁰ is a mature and well documented JavaScript framework which implements support for nested forms, editable lists and validation on top of the MVC framework BackboneJS²¹. Vie.js is JavaScript library for building *decoupled Content Management Systems*, and implements a bridge between Backbone.js and Semantic Web data. Vie also features a form generation component²². However, extending Backbone Forms with custom widgets requires hooking into the framework, such as by implementing certain interfaces. In contrast, this is not needed with the REX system, because with AngularJS custom widgets are used just like native HTML.

²⁰<https://github.com/powmedia/backbone-forms>

²¹<http://backbonejs.org/>

²²<http://viejs.org/widgets/forms/>

5 Conclusions and Future Work

In this deliverable we presented the REX system for creating highly dynamic HTML forms for RDF data based on the AngularJS framework. We showed how the system is used to realize a number of use cases, such as prefilling out form fields, conditionally emitting data, converting values between the fields and the RDF graph, and creating SPARQL 1.1 Update requests for persisting changes. We showed how our system enables idiomatic re-use of existing widgets, such as the date picker of AngularUI Bootstrap, and we created widgets for editing RDF terms and geometries ourselves. Future work will be to add rex attributes for filtering objects by language and datatype. This would enable only showing those fields whose language tags match given language settings. Further, investigating the use of REX for form generation approaches, which could arrange predefined HTML building blocks to create appropriate forms for RDF resources seems worthwhile. Also, the creation of an RDF vocabulary for abstract form descriptions which can then be processed by a form generator might be advantageous. Finally, there are several possibilities of combining this work with other efforts, such as the RDF Changeset systems developed in D4.3.1 or access control approaches in general.

6 Licence

Our newly developed RDF editing system and the Jassa library of D4.1.1 are freely available under the Apache v2 license²³. Note, that different licences may apply to third party components. For example, the widgets provided by AngularUI bootstrap²⁴ are under the MIT licence²⁵.

²³<http://www.apache.org/licenses/LICENSE-2.0.html>

²⁴<http://angular-ui.github.io/bootstrap/>

²⁵<https://github.com/angular-ui/bootstrap/blob/master/LICENSE>

Appendices

A Rex Directive Reference Table

Note: The *italic* items refer to the corresponding variables that will be made available in the scope of the element in **bold**.

Directive	Priority	Description
rex-iri="model"	1050	Convenience directive. Replaces itself with rex-object , rex-type="iri" and rex-value="model" .
rex-literal="model"	1050	Convenience directive. Replaces itself with rex-object , rex-type="literal" and rex-value="model" . By default a plain literal with no language tag is assumed. By using rex-lang or rex-datatype in addition, the literal can be augmented with a language or turned into a typed literal.
rex-typeof="model"	1050	Convenience directive. Replaces itself with rex-predicate="'rdf:type'" and rex-iri="model" .
ng-repeat	1000	Angular native directive listed for priority reference.
ng-init	450	Angular native directive listed for priority reference.
rex-context <i>rexContext</i>	30	A context is a container for triples generated by rex annotations on the same element or its children. The expression must evaluate to an object providing the RexContext interface.
rex-prefix <i>rexPrefix</i> <i>rexPrefixMapping</i>	27	Makes a given set of prefixes available in the current scope.
rex-lookup <i>rexLookup</i>	24	Declares a function in the current scope which instances of rex-subject will use to fetch RDF descriptions of referenced resources.
rex-subject <i>rexSubject</i>	21	This directive indicates a reference to a subject under a specified rex-context . The given argument must be evaluable to a string representing the IRI of the subject. If a lookup function was specified using rex-lookup , rex-subject will place the corresponding RDF triples into the <i>rexContext</i> .
rex-predicate <i>rexPredicate</i>	18	This directive indicates a reference to a predicate under a given rex-subject . The given argument must be evaluable to a string representing the IRI of the predicate.

Continued on next page

Table 1 – Continued from previous page

Directive	Priority	Description
rex-object <i>rexObject</i>	15	This directive indicates a reference to the <i>index</i> of an object under a predicate specified by rex-predicate . The given argument must be evaluable to an integer representing the index of the object. If no argument is provided, the value defaults to the first unused index under the parent rex-predicate directive, starting with 0.
<i>Component Reference Directives</i>		
rex-type <i>rexType</i>	12	This directive indicates a reference to the <i>type</i> component of an RDF term in regard to given rex-{subject, predicate, object} annotations. Valid values are: uri , literal or bnode . Note that blank nodes are not well supported by REX and their use is thus strongly discouraged.
rex-datatype <i>rexDatatype</i>	9	Analogous to rex-type , except that the <i>datatype</i> component is referenced.
rex-value <i>rexValue</i>	6	Analogous to rex-type , except that the <i>value</i> component is referenced.
rex-lang <i>rexLang</i>	3	Analogous to rex-type , except that the <i>lang</i> component is referenced.
ng-model	1	Angular native directive listed for priority reference.

B Synchronization Directives Reference Table

Directive	Priority	Description
<i>Convenience Directives</i>		
sync-template="templateString"	1050	Convenience attribute. Requires presence of an (ng-)model attribute. Replaces itself with sync-source="templateString" , sync-source-interpolate , sync-target="model" and sync-to-target .
sync-source="model"	n/a	This attribute is used to specify the source model of a sync connection.
sync-source-interpolate	n/a	The presence of this directive indicates that the source value should be considered a string for a template instead of a model expression. As such, Angular is instructed to process the value using the <i>\$interpolate</i> service, rather than <i>\$parse</i> . For example, using template strings such as <code>{{baseUri + givenName + lastName}}</code> require the presence of sync-source-interpolate .
sync-target="model"	n/a	This attribute is used to specify the target model of a sync connection.
sync-to-target="function"	0	The presence of this attribute causes the source model to be updated with the value of the target model whenever it changes. An optional transformation function taking the source value as the only argument and returning the target value can be provided.
sync-target-cond="model"	n/a	Attribute for specifying a condition when data is allowed to flow from source to target. When the outcome of the evaluation of the condition changes to true, the target is immediately updated with the corresponding source value, whereas if the condition changes to false no further update actions will occur. Requires sync-to-target .
sync-to-source="function"	0	Analogous to sync-to-target , except with source and target swapped. Cannot be used if sync-source-interpolate is present.
sync-to-source-cond="function"	n/a	Analogous to sync-to-target-cond , except with source and target swapped.