Implementing custom forms using N3 editing

Overview

The Vitro/VIVO system comes with basic RDF editing capabilities to add object and datatype statements to individuals. Frequently, people deploying Vitro /VIVO desire a web form which allows editing of multiple properties and individuals on the same form. A contact information form would be an example of a feature that would be implemented with a custom form in Vitro/VIVO.

The creation of a custom forms in Vitro/VIVO is done in two parts. The first is an implementation of the java interface EditConfigurationGenerator and the second is a FreeMarker template for the presentation. The EditConfigurationGenerator creates a EditConfiguration that controls how the values from the form will be used in the editing of the RDF, server side validation, which template to use, and other aspects of the edit. The FreeMarker template controls the HTML and Javascript for the form.

The EditConfigurationGenerator classes can be associated with an RDF property so that they are used from an individual's profile page or by a direct URL.

The main concept of custom forms is that the values submitted by the HTTP request will be substituted into placeholders in RDF N3 strings. These strings are then parsed to Jena RDF Model objects and that RDF is added to the system. For the modification of an existing value, a second set of strings is created and parsed which become the RDF statements to remove for the edit. This substitution is why the editing system is frequently called "N3 Editing". In practice, the N3 strings use only the turtle subset of the N3 syntax.

Steps of an Edit

Step 1. Getting the link to the edit

When a user is logged in, individual profile pages have edit links next to the listed properties. These links will take the user to a page with an edit form. The links on the individual profile page are routed to the EditRequestDispatchController which will determine which EditConfigurationGenerator to use based on which property is being edited. The VIVO/Vitro system can be configured to associate a EditConfigurationGenerator with a property so that the edit links will use a custom EditConfigurationGenerator. If no custom form is specified then the default object or data property EditConfigurationGenerator will be used.

A property can be associated with a custom form in one of two ways:
A) if you go to the site admin -object property hierarchy - the property you want associated with the form, click on the property then edit property record, you can put in the Java class name of the generator in the custom entry form field. E.g.edu.cornell.mannlib.vitro.webapp.edit.n3editing.configuration.generators.AddDistributionGenerator. This will allow you to associate the custom form while the system is running.

B) if you will be deploying the system for the first time and starting with an empty database, you would update vivo-core-1.5-annotations.rdf to specify that the property has a custom form using the vitro:customEntryFormAnnot property.

Step 2. Generating the EditConfiguration

When the user clicks the link, the client browser requests the URL of the edit link which will be to the EditRequestDispatchControl. That servlet will set up all that is needed in the session for the edit and respond with the HTML form. A custom form is setup by the EditConfigurationGenerator which has the sole purpose of making an EditConfiguration object. The EditRequestDispatchController will run getEditConfiguration() on the EditConfigurationGenerator to create the EditConfiguration. The EditConfiguration object has properties to define the characteristics of the edit. The EditConfiguration will specify the FreeMarker template for the form, and the server side instructions for validating the submitted result and instructions for processing the edit. When authoring a custom form, a central task is the coding of the EditConfigurationGenerator to produce an EditConfiguration that encodes logic of how you desire the edit to happen. The EditConfigurationGenerator is just a java class that creates an EditConfiguration.

When generating the EditConfiguration at runtime, an edit key will be created and the completed EditConfiguration will be associated with that key in the server side user session. This edit key is used to handle parallel editing and back button complexity. The EditConfiguration object for an edit is in a one to one relation with the HTML form for an edit. If the user goes to edit a street address and then goes to edit that street address a second time, the first edit will have an EditConfiguration object in the session and an HTML form with one edit key, and the second will have a different EditConfiguration in the session and an HTML form with a different edit key. An HTML form created for a edit will have an "edit key" to associate that specific instance of the HTML form with an object stored in the user's session.

The EditConfiguration can specify SPARQL queries for existing values for fields of the form. These are executed as part of the generation of the EditConfiguration.

Step 3. HTML creation by FreeMarker

Once the EditRequestDispatchController has the EditConfiguration and put it in the session, it will set up some standard values for the template and pass them and the EditConfiguration to the FreeMarker template specified in EditConfiguration.getTemplate(). The HTML form is then generated using the normal FreeMarker process. The HTML form must contain a field with the EditKey so associate the edit with an EditConfiguration in the session.

Step 4. Response From Client

The form will be submitted by the client's browser to ProcessRdfFormController. This will get the EditConfiguration based on the edit key from the submitted values. It will run validation and then substitute the values from the form into the N3 templates and parse the N3 to RDF. The N3 that gets created will be then added to the VIVO/Vitro models. If the edit is a change of an existing value, then the RDF for the statements to remove will be created and removed form the VIVO/Vitro models.