Linked (Open) Data

Introduction

Exchanging repository contents

Most sites on the Internet are oriented towards human consumption. While HTML may be a good format for presenting information to humans, it is not a good format to export data in a way easy for a computer to work with. Like most software for building repositories, DSpace supports OAI-PMH as an interface to expose the stored metadata. While OAI-PMH is well known in the field of repositories, it is rarely known elsewhere (e.g. Google retired its support for OAI-PMH in 2008). The Semantic Web is a generic approach to publish data on the Internet together with information about its semantics. Its application is not limited to repositories or libraries and it has a growing user base. RDF and SPARQL are W3C-released standards for publishing structured data on the web in a machine-readable way. The data stored in repositories is particularly suited for use in the Semantic Web, as the metadata are already available. It doesn't have to be generated or entered manually for publication as Linked Data. For most repositories, at least for Open Access repositories, it is quite important to share their stored content. Linked Data is a rather big chance for repositories to present their content in a way that can easily be accessed, interlinked and (re)used.

Terminology

We don't want to give a full introduction into the Semantic Web and its technologies here as this can be easily found in many places on the web. Nevertheless, we want to give a short glossary of the terms used most often in this context to make the following documentation more readable.

Semantic Web	The term "Semantic Web" refers to the part of the Internet containing Linked Data. Just like the World Wide Web, the Semantic Web is also woven together by links among the data.
Linked Data **Linked Open Data**	Data in RDF, following the Linked Data Principles are called Linked Data. The Linked Data Principles describe the expected behavior of data publishers who shall ensure that the published data are easy to find, easy to retrieve, can be linked easily and link to other data as well. Linked Open Data is Linked Data published under an open license. There is no technical difference between Linked Data and Linked Open Data (often abbreviated as LOD). It is only a question of the license used to publish it.
RDF **RDF/XML** **Turtle** **N-Triples** **N3-Notation**	RDF is an acronym for Resource Description Framework, a metadata model. Don't think of RDF as a format, as it is a model. Nevertheless, there are different formats to serialize data following RDF. RDF/XML, Turtle, N-Triples and N3-Notation are probably the most well-known formats to serialize data in RDF. While RDF/XML uses XML, Turtle, N-Triples and N3-Notation don't and they are easier for humans to read and write. When we use RDF in DSpace configuration files, we currently prefer Turtle (but the code should be able to deal with any serialization).
Triple Store	A triple store is a database to natively store data following the RDF model. Just as you have to provide a relational database for DSpace, you have to provide a Triple Store for DSpace if you want to use the LOD support.
SPARQL	The SPARQL Protocol and RDF Query Language is a family of protocols to query triple stores. Since version 1.1, SPARQL can be used to manipulate triple stores as well, to store, delete or update data in triple stores. DSpace uses SPARQL 1.1 Graph Store HTTP Protocol and SPARQL 1.1 Query Language to communicate with the Triple Store. The SPARQL 1.1 Query Language is often referred to simply as SPARQL, so expect the SPARQL 1.1 Query Language if no other specific protocol out of the SPARQL family is explicitly specified.
SPARQL endpoint	A SPARQL endpoint is a SPARQL interface of a triple store. Since SPARQL 1.1, a SPARQL endpoint can be either read-only, allowing o nly to query the stored data; or readable and writable, allowing to modify the stored data as well. When talking about a SPARQL endpoint without specifying which SPARQL protocol is used, an endpoint supporting SPARQL 1.1 Query Language is meant.

Linked (Open) Data Support within DSpace

Starting with DSpace 5.0, DSpace provides support for publishing stored contents in form of Linked (Open) Data.

Architecture / Concept

To publish content stored in DSpace as Linked (Open) Data, the data have to be converted into RDF. The conversion into RDF has to be configurable as different DSpace instances may use different metadata schemata, different persistent identifiers (DOI, Handle, ...) and so on. Depending on the content to convert, configuration and other parameters, conversion may be time-intensive and impact performance. Content of repositories is much more often read then created, deleted or changed because the main goal of repositories is to safely store their contents. For this reason, the content stored within DSpace is converted and stored in a triple store immediately after it is created or updated. The triple store serves as a cache and provides a SPARQL endpoint to make the converted data accessible using SPARQL. The conversion is triggered automatically by the DSpace event system and can be started manually using the command line interface – both cases are documented below. There is no need to backup the triple store, as all data stored in the triple store can be recreated from the contents stored elsewhere in DSpace (in the assetstore(s) and the database). Beside the SPARQL endpoint, the data should be published as RDF serialization as well. With dspace-rdf DSpace offers a module that loads converted data from the triple store and provides it as an RDF serialization. It currently supports RDF/XML, Turtle and N-Triples.

Repositories use Persistent Identifiers to make content citable and to address content. Following the Linked Data Principles, DSpace uses a Persistent Identifier in the form of HTTP(S) URIs, converting a Handle to http://hdl.handle.net/<handle> and a DOI to http://dx.doi.org/<doi>. Altogether, DSpace Linke d Data support spans all three Layers: the storage layer with a triple store, the business logic with classes to convert stored contents into RDF, and the application layer with a module to publish RDF serializations. Just like DSpace allows you to choose Oracle or Postgresql as the relational database, you may choose between different triple stores. The only requirements are that the triple store must support SPARQL 1.1 Query Language and SPARQL 1.1 Graph Store HTTP Protocol which DSpace uses to store, update, delete and load converted data in/out of the triple store and uses the triple store to provide the data over a SPARQL endpoint.

Store public data only in the triple store!

The triple store should contain only data that are public, because the DSpace access restrictions won't affect the SPARQL endpoint. For this reason, DSpace converts only archived, discoverable (non-private) Items, Collections and Communities which are readable for anonymous users. Please consider this while configuring and/or extending DSpace Linked Data support.

The org.dspace.rdf.conversion package contains the classes used to convert the repository content to RDF. The conversion itself is done by plugins. The or g.dspace.rdf.conversion.ConverterPlugin interface is really simple, so take a look at it you if can program in Java and want to extend the conversion. The only thing important is that plugins must only create RDF that can be made publicly available, as the triple store provides it using a sparql endpoint for which the DSpace access restrictions do not apply. Plugins converting metadata should check whether a specific metadata field needs to be protected or not (see org.dspace.app.util.MetadataExposure on how to check that). The MetadataConverterPlugin is heavily configurable (see below) and is used to convert the metadata of Items. The StaticDSOConverterPlugin can be used to add static RDF Triples (see below). The SimpleDSORelationsConverterPlugi creates links between items and collections, collections and communities, subcommunitites and their parents, and between top-level communities and the information representing the repository itself.

As different repositories uses different persistent identifiers to address their content, different algorithms to create URIs used within the converted data can be implemented. Currently HTTP(S) URIs of the repository (called local URIs), Handles and DOIs can be used. See the configuration part of this document for further information. If you want to add another algorithm, take a look at the org.dspace.rdf.storage.URIGenerator interface.

Install a Triple Store

In addition to a normal DSpace installation you have to install a triple store. You can use any triple store that supports SPARQL 1.1 Query Language and SPARQL 1.1 Graph Store HTTP Protocol. If you do not have one yet, you can use Apache Fuseki. Download Fuseki from its official download page and unpack the downloaded archive. The archive contains several scripts to start Fuseki. Use the start script appropriate to the OS of your choice with the options '--localhost --config=<dspace-install>/config/modules/rdf/fuseki-assembler.ttl'. Instead of changing to the directory into which you unpacked Fuseki, you may set the variable FUSEKI_HOME. If you're using Linux and bash, you unpacked Fuseki to /usr/local/jena-fuseki-1.0.1 and you installed DSpace to [dspace-install], this would look like this:

```
export FUSEKI_HOME=/usr/local/jena-fuseki-1.0.1 ; $FUSKI_HOME/fuseki-server --localhost --config [dspace-
install]/config/modules/rdf/fuseki-assembler.ttl
```

Fuseki's archive contains a script to start Fuseki automatically at startup as well.

Make Fuseki connect to localhost only, by using the argument --localhost when launching if you use the configuration provided with DSpace! The configuration contains a writeable SPARQL endpoint that allows any connection to change/delete the content of your triple store.
Use Apache mod proxy, mod rewrite or any other appropriate web server/proxy to make localhost:3030/dspace/sparql readable from the internet. Use the address under which it is accessible as the address of your public sparql endpoint (see the property public.sparql.endpoint in the configuration reference below.).

The configuration provided within DSpace makes it store the files for the triple store under [dspace-install]/triplestore. Using this configuration, Fuseki provides three SPARQL endpoints: two read-only endpoints and one that can be used to change the data of the triple store. **You should not use this configuration if you let Fuseki connect to the internet directly** as it would make it possible for anyone to delete, change or add information to the triple store. The option --localhost tells Fuseki to listen only on the loopback device. You can use Apache mod_proxy or any other web or proxy server to make the read-only SPARQL endpoint accessible from the internet. With the configuration described, Fueski listens to the port 3030 using HTTP. Using the address *http://localhost:3030/* you can connect to the Fuseki Web UI. *http://localhost:3030/dspace/data* addresses a writeable SPARQL 1.1 HTTP Graph Store Protocol endpoint, and *http://localhost:3030/dspace/get* a read-only one. Under *http://localhost:3030/dspace/sparql* a read-only SPARQL 1.1 Query Language endpoint can be found. **The first one of these endpoints must be not accessible by the internet**, while the last one should be accessible publicly.

Default configuration and what you should change

First, you'll want to ensure the Linked Data endpoint is enabled/configured. In your `local.cfg`, add `rdf.enabled = true`. You can optionally change it's path by setting `rdf.path` (it defaults to "rdf" which means the Linked Data endpoint is available at `[dspace.server.url]/rdf/` (where `dspace.server.url` is also specified in your `local.cfg`)

In the file `[dspace]/config/dspace.cfg` you should look for the property `event.dispatcher.default.consumers` and add `rdf` there. Adding rdf there makes DSpace update the triple store automatically as the publicly available content of the repository changes.

As the Linked Data support of DSpace is highly configurable this section gives a short list of things you probably want to configure before using it. Below you can find more information on what is possible to configure.

In the file `[dspace]/config/modules/rdf.cfg` you want to configure the address of the public sparql endpoint and the address of the writable endpoint DSpace use to connect to the triple store (the properties `rdf.public.sparql.endpoint`, `rdf.storage.graphstore.endpoint`). In the same file you want to configure the URL that addresses the dspace-rdf module which is depending on where you deployed it (property `rdf.contextPath`) and switch content negotiation on (set property `rdf.contentNegotiation.enable = true`).

In the file `[dspace]/config/modules/rdf/constant-data-general.ttl` you should change the links to the Web UI of the repository and the public readable SPARQL endpoint. The URL of the public SPARQL endpoint should point to a URL that is proxied by a webserver to the Triple Store. See the section Install a Triple Store above for further information.

In the file `[dspace]/config/modules/rdf/constant-data-site.ttl` you may add any triples that should be added to the description of the repository itself.

If you want to change the way the metadata fields are converted, take a look into the file `[dspace]/config/modules/rdf/metadata-rdf-mapping.ttl`. This is also the place to add information on how to map metadata fields that you added to DSpace. There is already a quite acceptable default configuration for the metadata fields which DSpace supports out of the box. If you want to use some specific prefixes in RDF serializations that support prefixes, you have to edit `[dspace]onfig/modules/rdf/metadata-prefixes.ttl`.

Configuration Reference

There are several configuration files to configure DSpace's LOD support. The main configuration file can be found under `[dspace-source]/dspace/config/modules/rdf.cfg`. Within DSpace we use Spring to define which classes to load. For DSpace's LOD support this is done within `[dspace-source]/dspace/config/spring/api/rdf.xml`. All other configuration files are positioned in the directory `[dspace-source]/dspace/config/modules/rdf/`. Configurations in `rdf.cfg` can be modified directly, or overridden via your `local.cfg` config file (see Configuration Reference). You'll have to configure where to find and how to connect to the triple store. You may configure how to generate URIs to be used within the generated Linked Data and how to convert the contents stored in DSpace into RDF. We will guide you through the configuration file by file.

[dspace-source]/dspace/config/modules/rdf.cfg

Property:	rdf.enabled
Example Value:	rdf.enabled = true
Informational Note:	Defines whether the RDF endpoint is enabled or disabled (disabled by default). If enabled, the RDF endpoint is available at ${dspace.server.url}/${rdf.path}. Changing this value requires rebooting your servlet container (e.g. Tomcat)
Property:	rdf.path
Example Value:	rdf.path = rdf
Informational Note:	Defines the path of the RDF endpoint, if enabled. For example, a value of "rdf" (the default) means the RDF interface/endpoint is available at ${dspace.server.url}/rdf (e.g. if "dspace.server.url = http://localhost:8080/server", then it'd be available at "http://localhost:8080/server/rdf". Changing this value requires rebooting your servlet container (e.g. Tomcat)

Property:	rdf.contentNegotiation.enable
Example Value:	rdf.contentNegotiation.enable = true
Informational Note:	Defines whether content negotiation should be activated. Set this true, if you use Linked Data support.
Property:	rdf.contextPath
Example Value:	rdf.contextPath = ${dspace.baseUrl}/rdf
Informational Note:	The content negotiation needs to know where to refer if anyone asks for RDF serializations of content stored within DSpace. This property sets the URL where the dspace-rdf module can be reached on the Internet (depending on how you deployed it).
Property:	rdf.public.sparql.endpoint
Example Value:	rdf.public.sparql.endpoint = http://${dspace.baseUrl}/sparql
Informational Note:	Address of the read-only public SPARQL endpoint supporting SPARQL 1.1 Query Language.
Property:	rdf.storage.graphstore.endpoint
Example Value:	rdf.storage.graphstore.endpoint = http://localhost:3030/dspace/data
Informational Note:	Address of a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint. This address is used to create, update and delete converted data in the triple store. If you use Fuseki with the configuration provided as part of DSpace 5, you can leave this as it is. If you use another Triple Store or configure Fuseki on your own, change this property to point to a writeable SPARQL endpoint supporting the SPARQL 1.1 Graph Store HTTP Protocol.

Property:	rdf.storage.graphstore.authentication
Example Value:	rdf.storage.graphstore.authentication = no
Informational Note:	Defines whether to use HTTP Basic authentication to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.
Properties:	rdf.storage.graphstore.login rdf.storage.graphstore.password
Example Values:	rdf.storage.graphstore.login = dspace rdf.storage.graphstore.password =ecapsd
Informational Note:	Credentials for the HTTP Basic authentication if it is necessary to connect to the writable SPARQL 1.1 Graph Store HTTP Protocol endpoint.
Property:	rdf.storage.sparql.endpoint
Example Value:	rdf.storage.sparql.endpoint = http://localhost:3030/dspace/sparql
Informational Note:	Besides a writable SPARQL 1.1 Graph Store HTTP Protocol endpoint, DSpace needs a SPARQL 1.1 Query Language endpoint, which can be read-only. This property allows you to set an address to be used to connect to such a SPARQL endpoint. If you leave this property empty the property ${rdf.public.sparql.endpoint} will be used instead.
Properties:	rdf.storage.sparql.authentication rdf.storage.sparql.login rdf.storage.sparql.password
Example Values:	rdf.storage.sparql.authentication = yes rdf.storage.sparql.login = dspace rdf.storage.sparql.password = ecapsd
Informational Note:	As for the SPARQL 1.1 Graph Store HTTP Protocol you can configure DSpace to use HTTP Basic authentication to authenticate against the (read-only) SPARQL 1.1 Query Language endpoint.

Property:	rdf.converter.DSOtypes
Example Value:	rdf.converter.DSOtypes = SITE, COMMUNITY, COLLECTION, ITEM
Informational Note:	Define which kind of DSpaceObjects should be converted. Bundles and Bitstreams will be converted as part of the Item they belong to. Don't add EPersons here unless you really know what you are doing. All converted data is stored in the triple store that provides a publicly readable SPARQL endpoint. So all data converted into RDF is exposed publicly. Every DSO type you add here must have an HTTP URI to be referenced in the generated RDF, which is another reason not to add EPersons here currently.
The following properties configure the StaticDSOConverterPlugin.	
Properties:	rdf.constant.data.GENERAL rdf.constant.data.COLLECTION rdf.constant.data.COMMUNITY rdf.constant.data.ITEM rdf.constant.data.SITE
Example Values:	rdf.constant.data.GENERAL = ${dspace.dir}/config/modules/rdf/constant-data-general.ttl rdf.constant.data.COLLECTION = ${dspace.dir}/config/modules/rdf/constant-data-collection.ttl rdf.constant.data.COMMUNITY = ${dspace.dir}/config/modules/rdf/constant-data-community.ttl rdf.constant.data.ITEM = ${dspace.dir}/config/modules/rdf/constant-data-item.ttl rdf.constant.data.SITE = ${dspace.dir}/config/modules/rdf/constant-data-site.ttl
Informational Note:	These properties define files to read static data from. These data should be in RDF, and by default Turtle is used as serialization. The data in the file referenced by the property ${rdf.constant.data.GENERAL} will be included in every Entity that is converted to RDF. E.g. it can be used to point to the address of the public readable SPARQL endpoint or may contain the name of the institution running DSpace. The other properties define files that will be included if a DSpace Object of the specified type (collection, community, item or site) is converted. This makes it possible to add static content to every Item, every Collection, ...
The following properties configure the MetadataConverterPlugin.	
Property:	rdf.metadata.mappings
Example Value:	rdf.metadata.mappings = ${dspace.dir}/config/modules/rdf/metadata-rdf-mapping.ttl
Informational Note:	Defines the file that contains the mappings for the MetadataConverterPlugin. See below the description of the configuration file [dspace-source] /dspace/config/modules/rdf/metadata-rdf-mapping.ttl.
Property:	rdf.metadata.schema
Example Value:	rdf.metadata.schema = file://${dspace.dir}/config/modules/rdf/metadata-rdf-schema.ttl

Informational Note:	Configures the URL used to load the RDF Schema of the DSpace Metadata RDF mapping Vocabulary. Using a file:// URI makes it possible to convert DSpace content without having an internet connection. The version of the schema has to be the right one for the used code. In DSpace 5.0 we use the version 0.2.0. This Schema can be found here as well: http://digital-repositories.org/ontologies/dspace-metadata-mapping/0.2.0. The newest version of the Schema can be found here: http://digital-repositories.org/ontologies/dspace-metadata-mapping/.
Property:	rdf.metadata.prefixes
Example Value:	rdf.metadata.prefixes = ${dspace.dir}/config/modules/rdf/metadata-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.
The following properties configure the SimpleDSORelationsConverterPlugin	
Property:	rdf.simplerelations.prefixes
Example Value:	rdf.simplerelations.prefixes = ${dspace.dir}/config/modules/rdf/simple-relations-prefixes.ttl
Informational Note:	If you want to use prefixes in RDF serializations that support prefixes, you can define these prefixes in the file referenced by this property.
Property:	rdf.simplerelations.site2community
Example Value:	rdf.simplerelations.site2community = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasCommunity
Informational Note:	Defines the predicates used to link from the data representing the whole repository to the top level communities. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2site
Example Value:	rdf.simplerelations.community2site = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfRepository

Informational Note:	Defines the predicates used to link from the top level communities to the data representing the whole repository. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2subcommunity
Example Value:	rdf.simplerelations.community2subcommunity = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasSubcommunity
Informational Note:	Defines the predicates used to link from communities to their subcommunities. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.subcommunity2community
Example Value:	rdf.simplerelations.subcommunity2community = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isSubcommunityOf
Informational Note:	Defines the predicates used to link from subcommunities to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.community2collection
Example Value:	rdf.simplerelations.community2collection = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasCollection
Informational Note:	Defines the predicates used to link from communities to their collections. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.collection2community
Example Value:	rdf.simplerelations.collection2community = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCommunity

Informational Note:	Defines the predicates used to link from collections to the communities they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.collection2item
Example Value:	rdf.simplerelations.collection2item = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasItem
Informational Note:	Defines the predicates used to link from collections to their items. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.item2collection
Example Value:	rdf.simplerelations.item2collection = http://purl.org/dc/terms/isPartOf, http://digital-repositories.org/ontologies/dspace/0.1.0#isPartOfCollection
Informational Note:	Defines the predicates used to link from items to the collections they belong to. Defining multiple predicates separated by commas will result in multiple triples.
Property:	rdf.simplerelations.item2bitstream
Example Value:	rdf.simplerelations.item2bitstream = http://purl.org/dc/terms/hasPart, http://digital-repositories.org/ontologies/dspace/0.1.0#hasBitstream
Informational Note:	Defines the predicates used to link from item to their bitstreams. Defining multiple predicates separated by commas will result in multiple triples.

[dspace-source]/dspace/config/modules/rdf/constant-data-*.ttl

As described in the documentation of the configuration file [dspace-source]/dspace/config/modules/rdf.cfg, the constant-data-*.ttl files can be used to add static RDF to the converted data. The data are written in Turtle, but if you change the file suffix (and the path to find the files in rdf.cfg) you can use any other RDF serialization you like to. You can use this, for example, to add a link to the public readable SPARQL endpoint, add a link to the repository homepage, or add a triple to every community or collection defining it as an entity of a specific type like a bibo:collection. The content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-general.ttl will be added to every DSpaceObject that is converted. The content of the file [dspace-source] /dspace/config/modules/rdf/constant-data-community.ttl to every community, the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-collection.ttl to every collection and the content of the file [dspace-source]/dspace/config/modules/rdf/constant-data-item.ttl to every Item. You can use the file [dspace-source]/dspace/config/modules/rdf/constant-data-site.ttl to specify data representing the whole repository.

[dspace-source]/dspace/config/modules/rdf/metadata-rdf-mapping.ttl

This file should contain several metadata mappings. A metadata mapping defines how to map a specific metadata field within DSpace to a triple that will be added to the converted data. The MetadataConverterPlugin uses these metadata mappings to convert the metadata of a item into RDF. For every metadata field and value it looks if any of the specified mappings matches. If one does, the plugin creates the specified triple and adds it to the converted data. In the file you'll find a lot of examples on how to define such a mapping.

For every mapping a metadata field name has to be specified, e.g. dc.title, dc.identifier.uri. In addition you can specify a condition that is matched against the field's value. The condition is specified as a regular expression (using the syntax of the java class java.util.regex.Pattern). If a condition is defined, the mapping will be used only on fields those values which are matched by the regex defined as condition.

The triple to create by a mapping is specified using reified RDF statements. The DSpace Metadata RDF Mapping Vocabulary defines some placeholders that can be used. The most important placeholder is dm:DSpaceObjectIRI which is replaced by the URI used to identify the entity being converted to RDF. That means if a specific Item is converted the URI used to address this Item in RDF will be used instead of dm:DSpaceObjectIRI. There are three placeholders that allow reuse of the value of a meta data field. dm:DSpaceValue will be replace by the value as it is. dm:LiteralGenerator allows one to specify a regex and replacement string for it (see the syntax of the java classes java.util.regex.Pattern and java.util.regex.Matcher) and creates a Literal out of the field value using the regex and the replacement string. dm:ResourceGenerator does the same as dm:LiteralGenerator but it generates a HTTP (S) URI that is used in place. So you can use the resource generator to generate URIs containing modified field values (e.g. to link to classifications). If you know regular expressions and turtle, the syntax should be quite self explanatory.

[dspace-source]/dspace/config/modules/rdf/fuseki-assembler.ttl

This is a configuration for the triple store Fuseki of the Apache Jena project. You can find more information on the configuration it provides in the section Install a Triple Store above.

[dspace-source]/dspace/config/spring/api/rdf.xml

This file defines which classes are loaded by DSpace to provide the RDF functionality. There are two things you might want to change: the class that is responsible to generate the URIs to be used within the converted data, and the list of Plugins used during conversion. To change the class responsible for the URIs, change the following line:

```
<property name="generator" ref="org.dspace.rdf.storage.LocalURIGenerator"/>
```

This line defines how URIs should be generated, to be used within the converted data. The LocalURIGenerator generates URIs using the ${dspace.url} property. The HandleURIGenerator uses handles in form of HTTP URLs. It uses the property ${handle.canonical.prefix} to convert handles into HTTPS URLs. The class org.dspace.rdf.storage.DOIURIGenerator uses DOIs in the form of HTTP URLs if possible, or local URIs if there are no DOIs. It uses the DOI resolver "http://dx.doi.org" to convert DOIs into HTTP URLs. The class org.dspace.rdf.storage.DOIHandleGenerator does the same but uses Handles as fallback if no DOI exists. The fallbacks are necessary as DOIs are currently used for Items only and not for Communities or Collections.

All plugins that are instantiated within the configuration file will automatically be used during the conversion. Per default the list looks like the following:

```
<!-- configure all plugins the converter should use. If you don't want to
         use a plugin, remove it here. -->
    <bean id="org.dspace.rdf.conversion.SimpleDSORelationsConverterPlugin" class="org.dspace.rdf.conversion.
SimpleDSORelationsConverterPlugin"/>
    <bean id="org.dspace.rdf.conversion.MetadataConverterPlugin" class="org.dspace.rdf.conversion.
MetadataConverterPlugin"/>
    <bean id="org.dspace.rdf.conversion.StaticDSOConverterPlugin" class="org.dspace.rdf.conversion.
StaticDSOConverterPlugin"/>
```

You can remove plugins if you don't want them. If you develop a new conversion plugin, you want to add its class to this list.

Maintenance

As described above you should add `rdf` to the property `event.dispatcher.default.consumers` and in dspace.cfg. This configures DSpace to automatically update the triple store every time the publicly available content of the repository is changed. Nevertheless there is a command line tool that gives you the possibility to update the content of the triple store. As the triple store is used as a cache only, you can delete its content and reindex it every time you think it is necessary of helpful. The command line tool can be started by the following command which will show its online help:

```
[dspace-install]/bin/dspace rdfizer --help
```

The online help should give you all necessary information. There are commands to delete one specific entity; to delete all information stored in the triple store; to convert one item, one collection or community (including all subcommunities, collections and items) or to convert the complete content of your repository. If you start using the Linked Open Data support on a repository that already contains content, you should run `[dspace-install]/bin/dspace rdfizer --convert-all` once.

Every time content of DSpace is converted or Linked Data is requested, DSpace will try to connect to the triple store. So ensure that it is running (as you do with e.g. your sevlet container or relational database).