REST API

DSpace REST API

This page is outdated, and refers to documentation for the 2011 Google Summer of Code prototype REST API project. This GSoC project is not developed anymore and it has spawned successors and alternatives. You can learn more about current REST API efforts here:

- Official REST API in DSpace 4.x (JERSEY based, Read Only)
- Official REST API in DSpace 5.x (JERSEY based, CRUD)
- Review of Existing Dspace REST API Frameworks
- DSpace Futures - REST API

This is a wiki page for DSpace REST API addon. The project is in development phase, but can be tested by the users. For the details please check this page.

Details

Project Title:	DSpace REST API
Author:	Bojan Suzic
Mentors (at GSoC):	Aaron Zeckoski, Mark Diggory
Contacting author:	*bojan.suzic* AT *gmail _DOT _com* - using subject line *DSpace REST*
SCM Location for Project:	http://scm.dspace.org/svn/repo/modules/dspace-rest/
Alternative (improved)	https://github.com/wijiti/dspace-rest-api

Project description

The REST approach promotes simplification and decoupling of software architecture, enabling further scalability, portability, granularity and simplified interaction of software systems and components.
The aim of this project is to provide DSpace with REST capable API and underlying component, which will enable developers and end-users to exploit the advantages of such approach.

Some of uses this module is intended to provide could be, for instance:

- interaction between DSpace systems and/or other repositories
- automation of different activities, e.g. submission of packages
- integrating repositories in process workflows of other applications or systems

- interaction with many kinds of systems or web applications, such as CMS, LMS, LCMS, VLS, AMS etc
- providing of other approaches to UI, such as client based/run UI
- crawling of repositories, exposing information in structural way

This project is continuation of GSoC 2009 and GSoC 2010 . In the first stage the basic support for REST for DSpace is provided, exposing many parts of DSpace functionality to the clients. Currently the module is being tested and improved in preparation for upcomign DSpace release.

Based on REST API for DSpace the new GSoC (2011) project is started. Vibhaj Rajan DSpace UI Client based on JavaScript and appropriate framework (JQuery). More information: Client UI REST API

Important: During Q1-Q3 of 2012 thanks to **Hayden Young** and other contributors from **Wijiti Pty Ltd** a great effort has been invested to continue the development of REST API for DSpace. They reworked the code base, refreshed the project to be compatible with newest DSpace 1.8.1 version, fixed the bugs and provided the integration of DSpace, REST API and Joomla CMS. There are also other numerous improvements introduced. **Their contribution can be reached via GitHub, including development site and documentation update.**

Based on this update, the integration with Joomla CMS is provieded as a part of the Saber Project, which has been implemented and demonstrated at Monash University.

Release plan and issues

It is expected to have working and tested code for DSpace 1.8 release. According to relevant discussions on DSpace developer meetings, there is possibility to have this code released asynchronously of DSpace, as independent module. After initial release the code will be actively maintained.

Currently (mid October 2010) there are several issues opened, of them I would notice the following:

- Performance issues during browsing bigger datasets. This is generally related to DSpace API. I have reported this problem and proposed solution at http://jira.dspace.org/jira/browse/DS-659. However as I am not sure whether it will be accepted and included in the upcoming release I will change the code and translate some functions used from DSpace API to DSpace REST API. This way some handling will be done directly at REST API level. Consequently some additional features related to sorting/ordering will be available.
- Multiple loops in listing Collections and some other entities. This issue is resolved as of end September. Additionally, option to fine-grain details level of the output is implemented (three levels).
- HTTP Basic Auth - this is easily to implement and will be done shortly
- Some Authorization related issues: the authorization handling is done on the level of DSpace API. However some functions translated to REST API are not directly available to users and thus do not provide Authorization mechanisms. In order to prevent misuse etc. this gap should be filled at REST API level.
- Not finished end-points. Working on them.
- Testing. Testing. I also need cooperation of you potential users. **I need repository for testing containing at least several hundreds of items. If you can provide me with that please contact me via email.**
- DSpace 1.5 and older versions support - planned to be implemented at the end of initial public release.

Recent changes

- June 10: XML input/digestion change, based on the input from Dhaivat Dave
- June 6: Added support for logo in communities and collections (issue reported by Vibhaj Rayan and Hayden Young)
- June 1: Applied Peter Dietz's patches (https://gist.github.com/952058) and pom.xml improvements discussed bellow on this page; fixed collection /items display reported by Hayden Young

Detailed activities

In the following sections main activities are elaborated in detail.

REST API Endpoints

In the following section listed are supported endpoints on the application level. **The items marked with dot (in C column) are in phase of implementation, while other items are considered already working.**

Please note that additional tests should be made in order to ensure proper stability of the whole application.

The sorting of the fields / output results is currently partially supported. This part of the application is implemented independently of the endpoints and will be worked on after the most of endpoints are completed.

Naming convention for endpoints

DSpace 1.x and 2.x are treating the resources on different way. 2.x is more generalized, suggesting the use of RDF-like interrelation notations.

Repository browsing

Earlier Implementation Description - GSoC09

C	Verb	URL	Description	Mandatory parameters	Optional parameters	Sorting fields	Response Data	Formats	Response codes
	GET	`/communities`	Returns a list of all communities on the system or return just top level communities.	-	`topLevelOnly=true` `idOnly=false` `detail=standard`	`id` `name` `countitems`	The list of communities containing respective fields . Response code details: `204` - if there are no communities on the system	`json` `xml`	200, 204, 400, 500
	GET	`/communities/{id}`	Return detailed information about `id` community.	`id`	`idOnly=false` `detail=standard`	-	Fields describing community.	`json` `xml`	200, 400, 404, 500

	GET	`/communities/{id}/{element}`	Return a particular data field found in the community `id`
			
			Fields supported (for `element`):
			`id` - entity identifier, internal to the system
			`name` - entity name
			`countItems` - number of items under community
			`handle` - handle of the community (unique persistent resource identifier)
			`type` - entity type (object type in the system)
			`collections` - collections contained in the community, ordered by id
			`canedit` - states user persmission on the community (editing)
			`anchestor` - anchestors of the community
			`children` - subcommunities, ordered by id
			`administrators` - group administrators, ordered by id
			`recent` - recent items in the community
			`shortDescription` - short description
			`copyrightText` - copyright text
			`sidebarText` - sidebar text
			`introductoryText` - introductory text
			`logo` - community logo; to retrieve use returned id with `/bistream/receive` endpoint
				`id`	`idOnly=false`
					
					`immediateOnly=true` `detail=standard`
						`id` `name` `countitems`	Respective field info	`json` `xml`	200, 204, 400, 500

	GET	`/collections`	Return a list of all collections in the system.	-	`idOnly=false` `isAuthorized=false` `detail=standard`	`id` `name` `countitems`	The list of the collections containing respective fields. Response code details: `204` - if there are no communities on the system	`json` `xml`	200, 204, 400, 500
	GET	`/collections/{id}`	Return detailed information about `id` collection	`id`	`idOnly=false` `detail=standard`	`id` `name` `countitems`	Fields of the collection entity.	`json` `xml`	200, 204, 400, 500
	GET	`/collections/{id}/{element}`	Return a particular data field found in the collection `id`. Fields supported (for `element`): `id` - entity identifier, internal to the system `name` - collection name `licence` - collection licence `items` - items contained in collection `handle` - handle of the collection (unique persistent resource identifier) `canedit` - states user permission on the collection (edit) `communities` - communities collection is a part of `countItems` - number of the items in the collection `type` - entity type (object type in the system) `shortDescription` - short description of the collection `introText` - introductory text for the collection `copyrightText` - copyright text for the collection `sidebarText` - sidebar text for the collection `provenance` - provenance `logo` - information about collection's logo (to retrieve use logo id in `/bitstream/receive` endpoint)	`id`	`idOnly=false` `immediateOnly=true` `#immediateonly` `detail=standard`	`id` `name` `countitems`	Respective field info	`json` `xml`	200, 204, 400, 500
	GET	`/items`	Return a list of the items in the system	-	`detail=standard`	-	The list of the items containing related fields . Response code details: `204` - if there are no communities on the system		

	GET	/items/{id}	Return detailed information about an item.	id	detail=standard	id name lastmodified submitter	Fields of the item entity.	json xml	200, 204, 400, 500

	GET	/items/{id}
		/{element}
			Return a particular data field fould in the item id
			
			Fields supported (for element):
			metadata - item metadata
			submitter - submitter group
			isArchived - archival status of the item
			isWithdrawn - states if the item is withdrawn
			owningCollection - owning collection of the item
			lastModified - last modified time
			collections - collections the item appears in
			communities - communities the item appears is
			name - name of the item
			bitstreams - bitstreams related to the item
			handle - item handle (unique identified)
			canedit - states can user edit the item
			id - item id
			type - element type
			bundles - bundles related to the item
				id, element	detail=standard	-	Respective field info	json
								xml
									200, 204, 400, 500

	GET	/bitstream/{id}	Return bitstream object - usually the library item file.	id	-	-	Fields of the bitstream entity.	json, xml	200, 400, 401, 403, 404, 500
	GET	/bitstream/{id}/{element	Return a particular data field found in bitstream id. Supported fields (for element): mimeType - mime type of file bundles - bundles the bitstream is a part of checkSum - checksum of the file checkSumAlgorithm - checksum algorithm used description - bitstream description formatDescription - file format description sequenceId - sequence id of the file size - size of the file source - source (typically filename with path information) storeNumber - asset store number where the bitstream is stored userFormatDescription - user's format description name - bitstream name handle - unique id of the bitstream id - internal id of the bitstream type - type of the entity (referring to bitstream) Note: bitstream can be not only the content of the item (like book pdf file etc), but also licence file or logo of community	id, element	detail=standard	-	Respective field info	json, xml	200, 400, 401, 403, 404, 500
	GET	/bitstream/{id} /receive	Return bitstream	id	-	-	Return bitstream	binary	200, 400, 401, 403, 404, 500
	GET	/groups	Return a list of the groups in the system	-	detail=standard	-	The list of the groups containing related fields . 204 if there are no groups in the system.	json,xml	200, 204, 400, 500
	GET	/groups/{id}	Return a group object	id	detail=standard	-	Fields of the group entity.	json,xml	200, 204, 400, 500
	GET	/groups/{id}/{element}	Return a particular data field found in the group entity id. Supported fields (for element): handle - unique id (external) id - internal id of the gruop isEmpty - is the group empty members - group members (as users) memberGroups - group members (as groups) name - group name type - entity type (referring to group)	id, element	detail=standard	-	Respective field info	json,xml	200, 204, 400, 500
	GET	/users	Return a list of the users in the system	-	detail=standard	-	The list of the users containing related fields .	json,xml	200,204,400,500
	GET	/users/{id}	Return a user info	id	detail=standard	-	Fields of the user entity.	json,xml	200,204,400,500

	GET	/users/{id}/{element}	Return a particular data field found in the user id . Supported fields (for element): email - user's email firstName - first name fullName - full name handle - handle (unique, external) id - internal id of the user language - preferred language lastName - last name name - name netId - network id requireCertificate - requires certificate to login selfRegistered - is user self registered type - type of the object	id,element	detail=standard	-	Respective field info	json,xml	200,204,400,500

Note: modifier idOnly is referred only to first layer of the results. For all other layers (e.g. nested results) only ids are returned in some cases, due to possible loops. Example: for community containing collections, on second level the response contains only ids for some elements where multiple loops may be created (community->has_collection->has_community....). Other data is modified according to idOnly flag.

Optional parameters

Parameter	Description
topLevelOnly	returns only top level communities
idOnly	if true return only the identifiers for the record
immediateOnly	return only direct parent community
isAuthorized	return only collections user has permission to work on
inArchive	return archived items for respective collection
detail	parameters: minimum, standard or extended control amount of details/deepthness exposed to user; e.g. should the sub-entities contain full descriptions

Sorting fields:

Not completed!

The sorting of the fields / output results is currently partially supported. This part of the application is implemented independently of the endpoints and will be worked on after the most of endpoints are completed.

Parameter	Description	Ordering supported
id	sort results by entity id	asc | ascending desc | descending
name	sort results by entity name	asc | ascending desc | descending
countitems	sort results by number of items contained	asc | ascending desc | descending
lastmodified	sort results by date of last item modification	asc | ascending desc | descending
submitterName	sort results by submitter name	asc | ascending desc | descending
submitterId	sort results by submitter id	asc | ascending desc | descending

Controlling results

Parameter	Description	Default	Example
_start	position of the first entity to return	0 (first)	_start=5 to list 6th item and onwards

`_page`	page of data to display	0 (first)	`_page=2`, to display second page with query results
`_perpage`	number of results to show on each page	0 (all)	`_perpage=10` to display 10 results per page
`_limit`	maximum number of entities to return	0 (all)	`_limit=50`

`_sort`		the sort order to return entities in
		should be comma separated list of field names
		suffix determines ordering
		suffixes: `_asc`, `_ascending`, `_desc`, `_descending`
		
		ascending default
			`sort=name`
			`_sort=name,email_desc,lastname_desc`

Response codes

Code	Description
200	OK
201	Created
204	No content
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
405	Method not allowed
500	Internal server error
503	Service unavailable

Authentication/Authorization

Currently only standard authentication is supported. The authentication data is provided in the request or header body.

Example:

```
/rest/communities.json?user=user@email.com&pass=userpassword
```

The same elements `user` and `pass` are used for header based authentication.

Authorization is done on underlying api level; in the case of error the proper message and error code are returned to the user.

Repository manipulation

C	Verb	URL	Description	Mandatory parameters	Optional parameters	Response Data	Formats	Response codes

	`POST`	`/communities`	Action to be done under community `id`, adding new content or values.
			
			Supported actions:
			`createAdministrators`
			`createCollection`
			`createSubcommunity`
				`id, action`
				
				-
				`name`
				`name`
					-	Id of newly created entity, depending on the action selected:
						id of group of administrators
						id of collection
						id of subcommunity
							`json`
							`xml`
								200, 400, 401, 403, 500

	PUT	/communities/{id}/{element}	Update the field `element` of the community `id`.
			
			Supported fields:
			`name` - change name
			`shortDescription` - change short description
			`copyrightText` - change copyright text
			`sidebarText` - change sidebar text
			`introductoryText` - change introductory text
			`collections` - add existing collection
			`children` - add existing subcommunity
				`id`
				
				`value`
				`value`
				`value`
				`value`
				`value`
				`cid`
				`cid`
					-	Response code		200, 400, 401, 403, 500

	`PUT`	`/communities/{id}/logo`	Set the logo for community `id`	`id`	-	Response code	binary	200, 400, 401, 403, 500
	`DELETE`	`/communities/{id}`	Delete community from the system	`id`	-	Response code	`json` `xml`	200, 400, 401, 403, 500

	DELETE	/communities/{id}/{element}/{eid}	Remove attribute/value eid of element element from the community id. Suported elements: collections children administrators	id, eid cid cid -	-	Response code	json xml	
	POST	/collections	Action to be done under collection id, adding new content or values. Supported actions: createAdministrators createSubmitters createTemplateItem createWorkflowGroup	id, action - - - step	-	Id ow newly created element	json xml	200, 400, 401, 403, 500
	PUT	/collections/{id}/{element}	Update field element of the collection id . Supported elements: shortDescription - short description introText - introductory text copyrightText - copyright text sidebarText - sidebar text provenance - provenance licence - collection licence name - collection name	id value value value value value value value	-	Response code	json xml	200, 400, 401, 403, 500
	DELETE	/collections/{id}	Delete collection from the system - CANNOT BE DONE DIRECTLY	-	-	Response code	json xml	200, 400, 401, 403, 500
	DELETE	/collections/{id}/{element}/{cid}	Remove attribute/value cid from collection id Supported attributes: administrators item submitters templateItem	id, cid	-	Response code	json xml	200,400,401,403,500
•	PUT	/collections/{id}/logo	Set the logo for collection id	id		Response code	binary	200,400,401,403,500
•	POST	/items	Action to be done under item id, adding content or value. Supported actions: createBundle createBitstream	id, action name name, input		Id of newly created element	json,xml, binary	200,400,401,403,500
•	PUT	/items/{id}/{element}	Update field element of the item id Supported fields: isArchived isWithdrawn owningCollection submitter	id,element	-	Response code	json,xml	200,400,401,403,500
•	DELETE	/items/{id}	Delete item from the system	id	-	Response code	json,xml	200,400,401,403,500
•	DELETE	/items/{id}/{element}/{eid}	Delete element/attribute eid from the item id Supported fields for element: bundle licences	id,element,eid	-	Response code	json,xml	200,400,401,403,500

Content searching

C	Verb	URL	Description	Mandatory parameters	Optional parameters	Sorting fields	Response Data	Formats	Response codes
	GET	/search	Return a list of all objects found by searching criteria. Notice: community and collection are mutually exclusive options.	-	modifiers{{query=query}}&(community=id or collection={{id}} idOnly=false	id name lastmodified submitter	Item info with basic metadata for the search results. Additionally return only identifiers when idOnly=true is used.	json xml	200, 204, 400, 500
	GET	/harvest	Return a list of all objects that have been created, modified or withdrawn within specified time range.	-	startdate {enddate}} community collection idOnly=false withdrawn=false Notice: community and collection are mutually exclusive options	-	Contains item info including id, name, handle, metadata, bitstreams according to the defined requirements. Additionally when idOnly=true only identifiers of results are returned. If the date is in incompatible format, error 400 is returned.	json xml	200, 204, 400, 500

Statistical info

C	Verb	URL	Description	Mandatory parameters	Optional parameters	Sorting fields	Response Data	Formats	Response codes
	`GET`	`/stats`	Return general statistics.	-	-	-	Cummulative list of statistics data for the system currently available.	`json` `xml`	200, 400, 500

Relationships interface

Experimental feature

This is considered as a experimental feature in the phase of being considered for compability with future versions of DSpace. Consider not important section; the status of the feature for upcoming release yet to be determined.

C	Verb	URL	Description	Mandatory parameters	Optional parameters	Sorting fields	Response Data	Formats	Response codes
•	GET	/resource/{handle}/relations	Return entities according to relation and parameters specified	`handle` `property`	`rtype` rfield	-	ontains entities selected and sorted in conformance to request parameters. For more details see description of `rtype` and`rfield`.	`json` `xml`	200, 204, 400, 401, 403, 500

Mandatory parameters

Parameter	Description	Values	Example

`property`	Return entities satisfying requested property relation	Structural properties
		`ds:isPartOfSite`
		`ds:isPartOfCommunity`
		`ds:isPartOfCollection`
		`ds:isPartOfItem`
		`ds:isPartOfBundle`
		`ds:hasCommunity`
		`ds:hasCollection`
		`ds:hasItem`
		`ds:hasBundle`
		`ds:hasBitstream`
		`ds:hasBitstreamFormat`
		
		Communities and collections
		`ds:logo`
		
		Bistream format
		`ds:support`
		`ds:fileExtension`
		`ds:mimeType`
		
		Bitstream
		`ds:messageDigest`
		`ds:messageDigestAlgorithm`
		`ds:messageDigestOriginator`
		`ds:size`
		
		Eperson
		`ds:language`
			`property=ds:hasCommunity` - return subcommunities of a community
			`property=ds:isPartOfCommunity` - return communities current community is part of (children)
			`property=ds:hasCollection` - return collections belonging to community
			`property=ds:hasItem` - return Items belonging to community

`rtype`	restriction on type - only entity with specifed type (s) would be returned	`ds:Bitstream`
		`ds:Bundle`
		`ds:Collection`
		`ds:Community`
		`ds:EPerson`
		`ds:Group`
		`ds:Item`
		`ds:DSpaceObject`
		`ds:Policy`
		`ds:Site`
		`ds:BitstreamFormat`|
			`rtype=ds:Collection` - return entities of Collection type

rfield	restriction on fields - return only selected fields; by default all fields are returned	id name countitems metadata subcommunities ancestors owner other (depending on object type, will be documented later)	rfield=id,name - return only entity id and name in response

Note: incomplete/orientative properties, for more info check [Vocabularies|http://code.google.com/p/dspace-sandbox/source/browse/#svn/modules/dspace-rdf/tags/dspace-rdf-1.5.1/src/main/java/org/dspace/adapters/rdf/vocabularies].

Visitors' suggestions or wishes

Here the visitors and stakeholders can insert their suggestions or describe the needs for their applications in detail.

Comment: In this case it is not clear how to treat `recent` part of endpoint. If we stick to semantic mapping, then it should look like `/resource/id/mapping`, but `recent` in this case obviously do not represent a mapping, but the property.

Comment #2: Semantic mapping presented in this case should be probably hardcoded for 1.x branch, but on abstraction level which enables easy replacement with some auto-discovery method prepared for 2.x and eventually backported to 1.x. This way we would be able to call something similar to `/communities/id` or `communities/id/capabilities` in order to get supported mappings (amongst other data).

Suggesting new options:

Instead of changing wiki contents visitors can enter their suggestions as a comments.

1) **Kevin S. Clarke and Tim Donhue** suggested adding of the new feature related to HTTP Basic Auth. Ok, I will investigate how it could be done and included here. More info comming.

Integration in the system

It is planned to consult two external subjects for cooperation and the assistance during integration process (LMS and national library internal automation process). More information coming soon - awaiting approval of other parties.

Documentation tasks

Although provided software module exposes basic documentation automatically to the end user, in order to make it easier for other developers and users the documentation in the following forms is additionaly to be provided:

- Confluence pages, current location
- integrated documentation in PDF form (manual)
- short slides containing technology overview, advocacy/facts, configuration and usage guideliens and examples
- code will be additionally commented

Example of usage

At the end of the current stage of this project as a bonus task (if time constraints allow) the examples of usage will be provided for several languages, the use-cases will be presented (example of integration in other software, e.g. LMS) and optionally simple client system demonstrating UI customization will be demonstrated (e.g. Flex or JavaFX like).

How to install and test the module?

Here I will show two ways to install and test this module.

Update: Current version includes this update (thanks Peter!). This explanation will be shortly removed.

~~Note: The code for the REST API from the Google Summer of Code 2010 may be out of date with the latest version of dspace. It may help to import the latest stable version of DSpace through the REST API pom.~~

~~To do so, modify [dspace-rest-api-source]/pom.xml~~

```
Index: pom.xml
===================================================================
--- pom.xml        (revision 6356)
+++ pom.xml        (working copy)
@@ -18,7 +18,7 @@
     <parent>
         <artifactId>dspace-parent</artifactId>
         <groupId>org.dspace</groupId>
-        <version>1.6.0-SNAPSHOT</version><!--dspace2.version-->
+        <version>1.7.1</version><!--dspace2.version-->
     </parent>

     <repositories>
@@ -100,7 +100,7 @@
             <groupId>org.dspace</groupId>
             <artifactId>dspace-api</artifactId>
             <scope>compile</scope>
-            <version>1.6.2-SNAPSHOT</version>
+            <version>1.7.1</version>
         </dependency>

         <dependency>
@@ -121,7 +121,7 @@
             <groupId>org.dspace</groupId>
             <artifactId>dspace-jspui-api</artifactId>
             <scope>compile</scope>
-            <version>1.6.2-SNAPSHOT</version>
+            <version>1.7.1</version>
             <exclusions>
                 <exclusion>
                     <groupId>org.dspace</groupId>
@@ -133,7 +133,7 @@
             <groupId>org.dspace</groupId>
             <artifactId>dspace-api-lang</artifactId>
             <scope>compile</scope>
-            <version>1.5.2.2-SNAPSHOT</version>
+            <version>1.7.1.0</version>
         </dependency>

         <!--
```

For both approaches you should have Apache Maven installed. Then proceed using Subversion and check out the code from http://scm.dspace.org/svn/repo/modules/dspace-rest/trunk

1) This way assumes you are running DSpace under Tomcat. Locate `src/main/webapp/WEB-INF/web.xml` (under directory you just downloaded DSpace REST API). Find variable named `dspace-config` and alter it to point to current location of `dspace.cfg` file of your DSpace instance. Navigate to the root directory of the REST API and type `mvn package`. If everything goes well, in `target` directory will be packaged `dspace-rest-[version].war`. Deploy this file (changing the name to rest.war) to your current Tomcat webapp directory. The application will be available under http://localhost:8080/rest/ by default.

2) You can run REST API under Jetty container. Proceed with the same steps as under #1. Then instead to deploy .war file to Tomcat web server, from the root of REST API source tree issue command `mvn jetty:run-war`. This will run REST support under Jetty and the web point will be available at http://localhost:8080/dspace-rest/ by default.

(Option 3) Install as a DSpace Module

If you have an existing instance of DSpace that you are developing, you can connect the rest api module to your existing code base by adding it as a module.

Modify [dspace-source]/dspace/pom.xml by adding the path to the checked out rest code.

```
--- a/dspace/pom.xml
+++ b/dspace/pom.xml
@@ -505,6 +505,7 @@
     -->
     <modules>
         <module>modules</module>
+       <module>../../../dspace-rest-gsoc10</module>
     </modules>

     <build>
```

Once you rebuild your dspace-src code with mvn package and ant update, you will additionally need to copy the compiled .war file produced in the dspace-rest-gsoc10 target directory to tomcat's webapps directory.

```
cp /path/to/dspace-rest-gsoc10/target/dspace.war /var/lib/tomcat6/webapps/rest.war
```

Afterwards you can restart tomcat and visit the rest api in action at: http://localhost:8080/rest

(Option 3a) Install as a DSpace Module in a source code installation

Its unlikely you will want to do this unless you are a committer or just nosey like me and like to play around with the code.

1) Create a new directory for the REST module source code - dspace-src/dspace-rest.

2) Checkout the source code from http://scm.dspace.org/svn/repo/modules/dspace-rest/trunk/ into the new directory.

3) Incorporate the new module into your project by adding a new <module> element for dspace-rest to the 'all' profile in dspace-src/pom.xml.

4) Tell Maven to use your new local module by adding a new <profile> to dspace-src/dspace/pom.xml. If you don't do this the project will build okay but won't be using your local source code for that module.

5) Create a new directory dspace-src/dspace/modules/rest.

5a) Add a sub-directory src/main/webapp and a pom.xml to the directory created in 5. (Copy the pom from any other modules/xxxx module).

6) Add a <profile> to dspace-src/dspace/modules/pom.xml.

7) Rebuild your project.

Possible problems:

- If you have trouble starting the application, check the `dspace-config` variable and make sure it points to the location of the `dspace.cfg` file. Use absolute addressing (see comment in src/main/webapp/WEB-INF/web.xml).

- If you receive HTTP 500 errors with a SQL exception indication *and* you are using Oracle, make sure you have ojdbc14.jar in your CLASSPATH when you start tomcat or jetty.

- If you are already running another application on port 8080 try instead to start Jetty container with the following line: `mvn jetty:run-war -Djetty.port=9090` for port 9090.

Please note this is still an experimental module so there may be bugs/errors in processing. Use it at your own risk.

I would highly appreciate user input. If you have comments or feature requests or anything else you can post it on this wiki in comments section. Additionally for the bugs/errors/issues found you can use JIRA at https://jira.duraspace.org/browse/DS/component/10190 to report or contact me directly via email (*bojan.suzic* AT *gmail _DOT _com* - using subject line *DSpace REST)*.

Information for developers

In this section the main sections of the software will be briefly explained in order to ease update or extension of the components.

The REST API for DSpace uses Aaron Zeckoski's EntityBus and DSpace standard libraries, as dspace-api.

There are two main packages: **org.dspace.rest.providers** and **org.dspace.rest.entities**. Providers are responsible for serving content/feeds to the users. They usually prepare entities or particular entity and/or handle update/delete/create functions. The main class there is **AbstractBaseProvider**, which is extended by other providers.

In the providers, at the constructor level created are mappings between particular endpoints (e.g. /rest/items/1/collections) and related functions in entities (org.dspace.entities.items.getCollections). Thus, for each GET, PUT, POST or DELETE function in the entity provider's constructor defined are such mappings between URL endpoints and functions. After the client makes specific call, the provider prepares answer and in this phase calls mapped function and prepare results for display.

So, if you want to extend currently available endpoints for already present providers and entities, it is necessary to define mapping at provider level and prepare corresponding function at the entity level (based on the template). The system then calls this function and provides necessary arguments for its successfully handling.

If you want to develop a new provider, it is usually necessary to create new provider class in org.dspace.rest.providers and then create related entity in org. dspace.rest.entities. The currently available providers are good example how to do that.