iCore
Empowering IoT through Cognitive Technologies

SEVENTH FRAMEWORK PROGRAMME

VO Registry API Developer's Cookbook

Version:	2.4.6
Total Number of Pages:	15
File:	VO Registry API Developer's Cookbook

Abstract

This document presents the overview of the implemented work with respect to the VO Registry concepts. In particular, it describes in details the Java API that is used for the creation of VO Description based on the VO Model, as well as the interfaces for the interaction with the available VO Registry instances.

Keywords List

VO Model, VO Semantics, Graph Database, meta-data, VO Registry API, VO Registry Interface, Java API.

Table of Contents

1. *Introduction*

The current implementation constitutes a Java implementation of the VO Registry API. The VO Registry API has a set of different specifications that are / possibly will be defined in the D3.3 iCore document. Based on these specifications this Java API is constructed so as to help the Java developers to implement their own custom algorithms for the interaction with the iCore VO Registry.
By using this API the developers have the ability to perform various processes that are summarized as follow:

1 Dynamic creation of the semantic description of the VOs and dynamic registration of the VO in the VO Registry.
2 Dynamic loading of the VO Description Templates contents in form of XML, JSON, RDF/XML, RDF/JSON and dynamic registration of the VO in the VO Registry.
3 Dynamic generation of SPARQL Requests by using the Java API Classes
4 Dynamic discovery of VOs by using specific search criteria. For the implementation of the discovery process it is used the SPARQL Language and by using the API it is able to generate SPARQL code, just by calling the corresponding java classes.
5 Dynamic modification of VOs and their properties by using SPARUL (aka SPARQL 1.1 Update) for the performance of Update & Delete requests toward the VO Registry. The corresponding SPARQL Update code is automatically generated by using the corresponding java classes.
6 Dynamic construction of VO Registry Requests based on the corresponding specifications.
7 Communication interface between the VO Registry and its clients, by using HTTP-REST, with the data to be exchanged in usual popular forms such as XML and JSON.

2. *VO Model Overview*

2.1 VO Model Specifications

The VO model has been developed so as to support the creation of the semantic description of the VOs. It is comprised of a set of different features that correspond to specific meta-data, which is defined into an OWL ontology.
Figure 1, depicts the current structure of the VO Model, which is developed as Graph composed by different linked nodes, where each node includes a set o different meta-data properties. The associations between the nodes (namely the arrows between the boxes) are characterized by semantically annotated properties which are called **'Predicates'** and define the sort of the association between the nodes. In addition the node where the arrow starts is called '**Subject'**, while the node where the arrow ends up is called **'Object'**. For example a VO has always an owner, usually the VO installer, which is uniquely identified by a specific URI. This could be expressed as follow: VO ---hasOwner ---> User_X, where VO: is the Subject, hasOwner: is the predicate and User_X: is the Object.

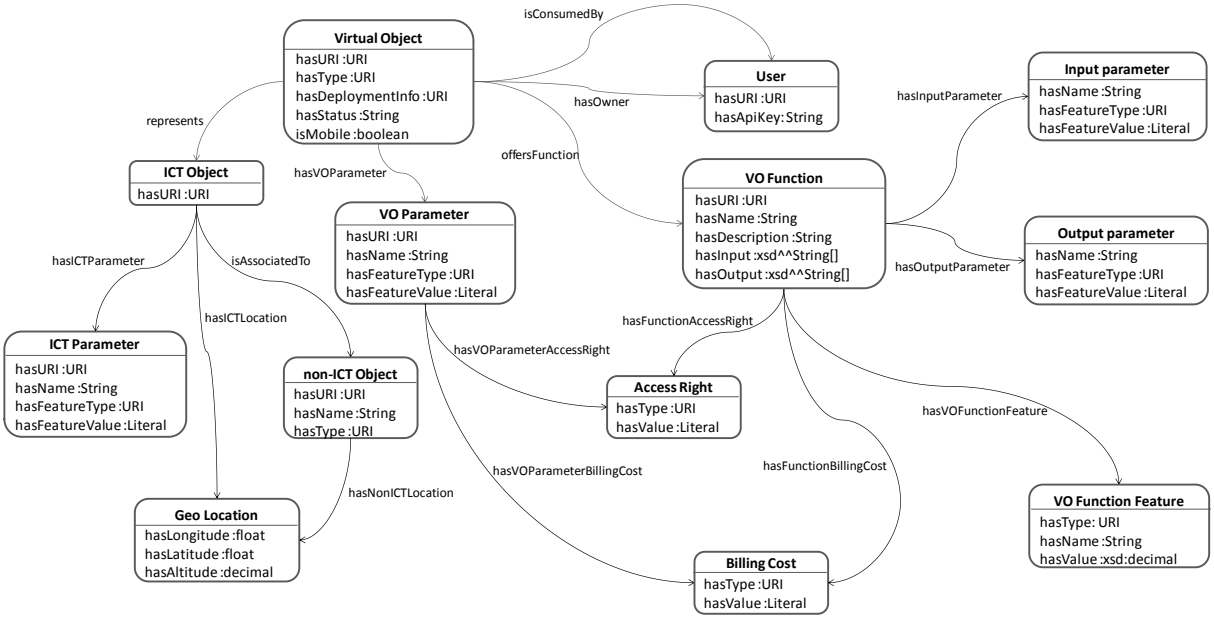

Figure 1: VO Model Graph

2.2 VO Model Java API – A reference Implementation

2.2.1 API Overview

Based on the VO Model, it has been developed a set of classes that support the dynamic creation of the VO Description and furthermore the dynamic registration of the VO by storing its description in the VO Registry Graph Database.

The dynamic creation of the VO Description through the use of the VO Registry Java API can be realized by two ways:

- by using the VO Description Templates that can be filled manually and can be developed in XML, JSO, RDF/XML and RDF/JSON,
- by using the appropriate Java classes and their methods, which are available in the **icore-voregistry-client-api-vX.Y.Z.jar** java library.

In the Java API the nodes and their properties that are included in the VO Model, they are consider as VO Model Variables. The Table 1 presents the description of the VO Model Variables.

VO Model Attribute (Named as it is in the Java API)	VO Model Properties (in the VO Model Attribute)	Description	Required
VirtualObejct	**URI** **Possible Values:** blank **/** URI **Java Methods:** VirtualObject()/VirtualObject (URI)	It should be unique. Currently it is given manually by the user **For the Butler VOs could be: http://purl.oclc.org/butler/VO/VO_ID	YES
	Type **Possible Values:** URI **Java Methods:** getType/setType	The type of the device or of the service that is Virtualized. **For example the Smart Server in case it is considered as Device will have the Type: http://purl.oclc.org/NET/ssnx/ssn#Device, while in case it is considered as Service it will have the Type:	

		http://purl.oclc.org/NET/ssnx/ssn#Service	
	DeploymentInfo **Possible Values:** URI **Java Methods:** getDeploymentInfo/setDeploymentInfo	This is the URL where will exist a file with deployment information for the VO (e.g.: a WSDL, Web.xml, etc) **(For now just you can leave it as it is in the java example, since it is PURL and don't affect the implantation).	
	Status **Possible Values:** VOStatus (AVAILABLE / UNAVAILABLE) **Java Methods:** getDeploymentInfo/setDeploymentInfo	Describes the availability of the VO. Due to the registration it is always AVAILABLE.	
VOOwner	**VOOwner** **Possible Values:** URI & String **Java Methods:** getVOOwner/setVOOwner	This attribute represents the VO Owner that essentially inherits the User class. In order to define the VO Owner should give the URI that corresponds to the User entity as well as the unique API-KEY that corresponds to the User.	YES
VOConsumer	**VOConsumer** **Possible Values:** URI & String **Java Methods:** getVOConsumersList / getVOConsumersList().add(VOConsumer)	This attribute corresponds to the end-users (both Humans / Software Agents) that consumes the VO. A particular example could be a CVO that has in its composition the VO. The structure of the VO Consumer includes the URI that correspond to the end-user and it API-KEY.	NO (in the initial registration of the VO)
VOParameter	URI **Possible Values:** automatically created by the constructor. **Java Methods:** VOParameter()	It can be used for the definition of specific features that regard the VO. A usual example for this case refers to the definition of the communication protocols that are used by the VO.	NO
	Name **Possible Values:** Java String, (e.g.: "Communication Protocol"). **Java Methods:** getName / setName		
VOParameter MetaFeatureSet	Type **Possible Values:** URI **Java Methods:** getType/setType	For the definition of this node can be used the FeatureType & FeatureValue pair of properties. The URI in the Type parameter corresponds to the URI of an ontology. Currently, the URIs are dummies and usually we are using the form: http://www.-iot-icore.eu/ontologies/ On the other hand the value can be literal.	NO
	Value **Possible Values:** Java Object **Java Methods:** getValue/setValue		

		For example in order to describe a VO that supports two communication protocols we could have the following: ---------- Name: Communication Protocol 1 Type: http://www.iot-icore.eu/ontologies/Protocols# Value: HTTP-REST ---------- ---------- Name: Communication Protocol 2 Type: http://www.iot-icore.eu/ontologies/Protocols# Value: MQTT ----------	
VOFunction	URI **Possible Values:** automatically created by the constructor. **Java Methods:** VOFunction()	This node with its properties constitutes the description of the VO Functionality.	YES
	Name **Possible Values:** String **Java Methods:** getName/setName		
	Description **Possible Values:** Java String **Java Methods:** getDescription/setDescription		
	Inputs Names **Possible Values:** HashSet<String> **Java Methods:** getInputNames/setInputNames		
	Outputs Names **Possible Values:** HashSet<String> **Java Methods:** getOutputNames/setOutputNames		
InputParameter	Name **Possible Values:** String **Java Methods:** getName/setName	The description of each available input parameter.	YES

InputParameter MetaFeatureSet	Type **Possible Values:** URI **Java Methods:** getType/setType Value **Possible Values:** Java Object **Java Methods:** getValue/setValue	For the definition of this node can be used the FeatureType & FeatureValue pair of properties. The URI in the Type parameter corresponds to the URI of an ontology. Currently, the URIs are dummy and usually we are using the form: http://www.-iot-icore.eu/ontologies/ On the other hand the value can be literal. For example in order to describe the input parameter can have as inputs decimals, we could have the following: ---------- Name: Data Type Type: http://www.w3.org/2001/XMLSchema# Value: Decimal ----------	NO
OutputParameter	Name **Possible Values:** String **Java Methods:** getName/setName	The description of each available output parameter.	YES
OutputParameter MetaFeatureSet	Type **Possible Values:** URI **Java Methods:** getType/setType Value **Possible Values:** Java Object **Java Methods:** getValue/setValue	For the definition of this node can be used the FeatureType & FeatureValue pair of properties. The URI in the Type parameter corresponds to the URI of an ontology. Currently, the URIs are dummies On the other hand the value can be literal. For example in order to describe the output parameter can return values that have as unit the cm, we could have the following: ---------- Name: Units of Measurement Type: http://purl.obolibrary.org/obo/uo.owl# Value: cm ----------	NO
VOFunctionFeature	URI **Possible Values:** automatically created by the constructor.	This node is used so as to describe the VO Function in terms of features with negative and positive meaning. *Positive meaning* → features of the type	YES

	Java Methods: VOFunctionFeature() Name **Possible Values:** String **Java Methods:** getName/setName Type **Possible Values:** URI VOFunctionFeatureType.COST VOFunctionFeatureType.UTILITY **Java Methods:** getType/setType Value **Possible Values:** String *If the Type is Cost*: VOFunctionFeatureCost.ENERGY VOFunctionFeatureCost.NETWORK VOFunctionFeatureCost.EXPENDITURE *If the Type is Utility*: VOFunctionFeatureUtility.QUALITY VOFunctionFeatureUtility.PERFORMANCE VOFunctionFeatureUtility.SECURITY **Java Methods:** getValue/setValue	**'Utility'** *Negative Meaning* → features of the Type '**Cost'.**	
BillingCosts	Type **Possible Values:** URI **Java Methods:** getType/setType Value **Possible Values:** Java Object **Java Methods:** getValue/setValue	This node describes the potential billing costs for the VO Function and/or a VO Parameter. The URI in the Type parameter corresponds to the URI of an ontology. Currently, the URIs are dummy and usually we are using the form: http://www.-iot-icore.eu/ontologies/ On the other hand the value can be literal. For example in order to describe the input parameter can have as inputs decimals, we could have the following: ---------- Name: Price per Function Call Type: http://www.-iot-icore.eu/ontologies/BillingCost#Price_per_Call Value: 0.01	NO

ICTObject	URI **Possible Values:** automatically created by the constructor. **Java Methods:** ICTObject()	The ICT Object the is represented by the VO, e.g.: sensor, actuator, device, smart phone, tag, etc.	YES
ICTParameter	URI **Possible Values:** automatically created by the constructor. **Java Methods:** ICTParameter()	Various parameters that can be used so as to describe specific features for the ICT Object, e.g. the technical specifications of a sensor.	NO
Geolocation	URI **Possible Values:** automatically created by the constructor. **Java Methods:** GeoLocation()	The Geolocation of the ICT Object as well as of the Non-ICT Object. Both ICT and Non-ICT Object can have the same or different location. Consequently, the constructor can be used for the creation of different instances that will be associated with the ICT and the Non-ICT object respectively.	YES
	Longitude **Possible Values:** java double **Java Methods:** getLongitude / setLongitude		
	Latitude **Possible Values:** java double **Java Methods:** getLatitude / setLatitude		
	Altitude **Possible Values:** java double **Java Methods:** getAltitude / setAltitude		
NonICTObject	URI **Possible Values:** automatically created by the constructor. **Java Methods:** NonICTObject()	This node refers to the Non-ICT object that is associated with the ICT object. The URI in the Type parameter corresponds to the URI of an ontology. Currently, the URIs are dummy and usually we are using the form: http://www.-iot-icore.eu/ontologies/ On the other hand the value can be literal. For example in order to describe the input parameter can have as inputs decimals, we could have the following: ---------- Name: Sarah's Home Type: http://www.-iot-icore.eu/ontologies/Places#Room Value: Kitchen ----------	YES
	Name **Possible Values:** String **Java Methods:** getName/setName		
	Type **Possible Values:** URI **Java Methods:** getType/setType		

Table 1: VO Model Variables

3. VO Registry Overview

3.1 VO Registry specifications

The Figure 2 presents an overview of the VO Registry design, while the Table 2 presents an overview of the implementation technologies that have been used for the development of the modules that belong to the VO Registry domain.

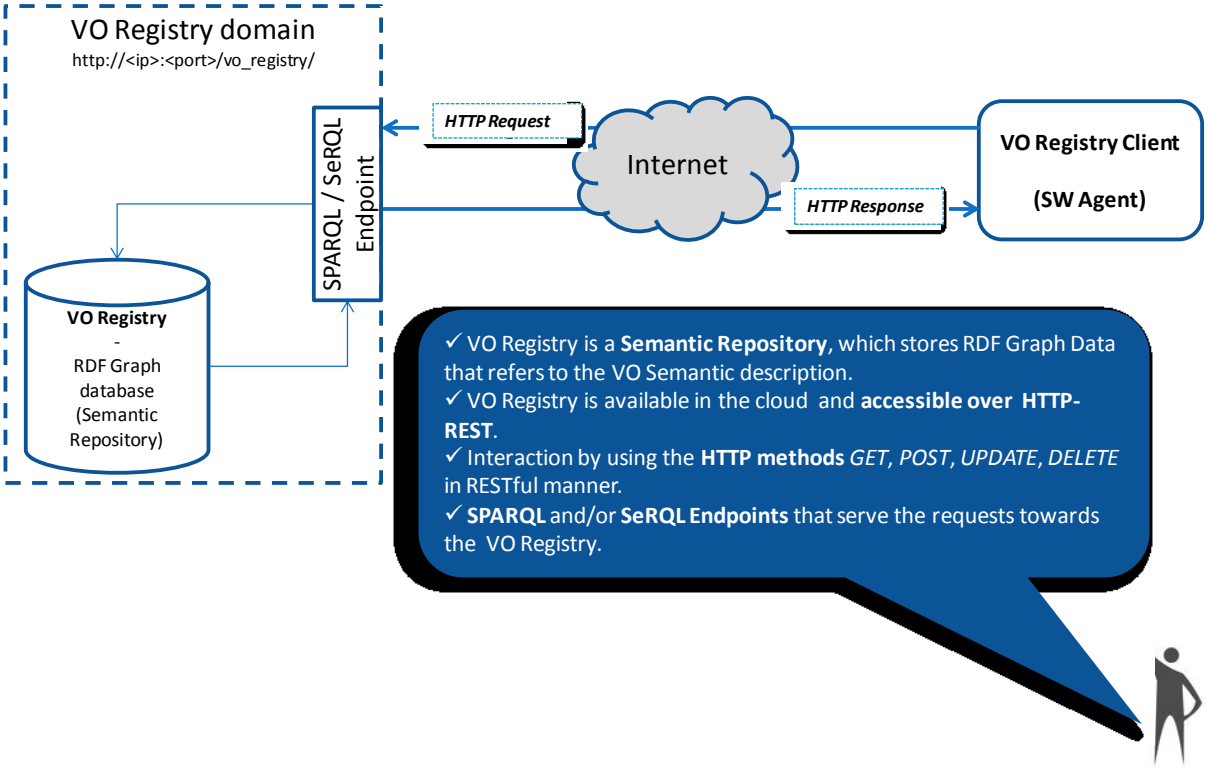

Figure 2: VO Registry Design overview

3.2 VO Registry proposed implementation Technologies

VO Registry Module/Entity		Implementation Technology	
Semantic Repository		openRDF Sesame API	
SPARQL Endpoint		Apache Jena API	
Communication Interface		HTTP REST	
VO Registry Host		Sesame Web Server & RESTY Simple Web Server	
VO Registry Client		REST Client	
Communication Scheme	VO Registry Request	VO Registration Request	(HTTP POST)
		VO Discovery Request	(HTTP GET)
		VO Update Request	(HTTP PUT)
		VO Delete Request	(HTTP DELETE)
		VO Registry Response	(HTTP Response)
		Data Format	XML, JSON, RDF/XML, RDF/JSON, N-Triples, Turtle

Table 2: VO Registry proposed implementation technologies

3.3 VO Registry Java API – A reference implementation

3.3.1 API Overview

The VO Registry Java API is comprised by 3 different parts; (a) the VO Model, (b) the VO Registry Client and (c) the VO Registry Server.

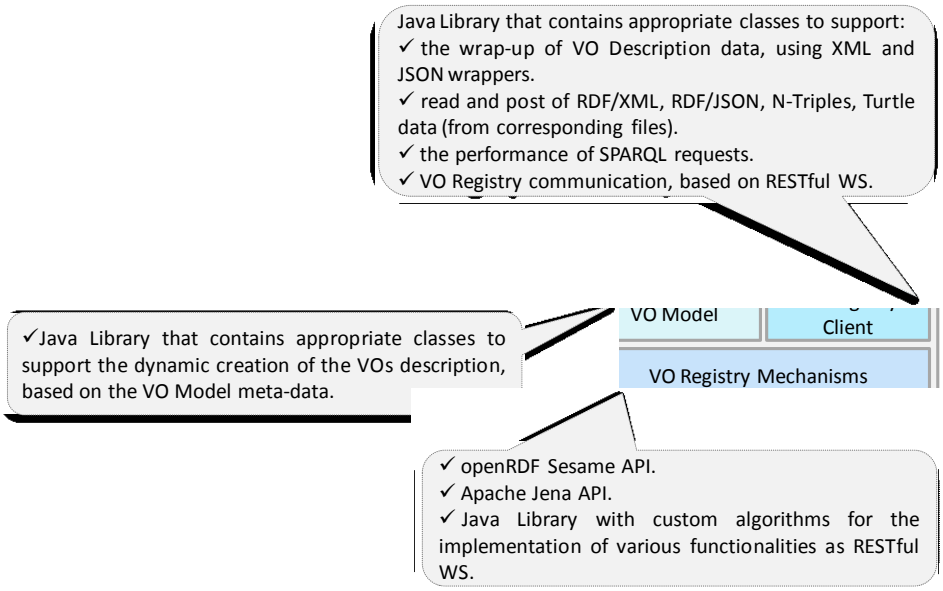

Figure 3: VO Registry Java API overview

The library that is shared includes the required modules for the VO Model and the Client development, while the VO Registry Server module is used only by the VO Registry owners, not by

the end-users. This happens because the end-user requirements / needs are fully covered by the VO Model and the VO Registry Client modules and the end-user either human or software agent has the absolute freedom to perform requests and get response from the VO Registry server side.

3.3.2 VO Registry Interface

The VO Registry interface includes two (2) types of data type; (a) the VO Registry Request (Figure 4) and (b) the VO Registry Response (Figure 5). In addition, the Table 3 and Table 4 present the description of the structure for the VO Registry Request and VO Registry Response, respectively.

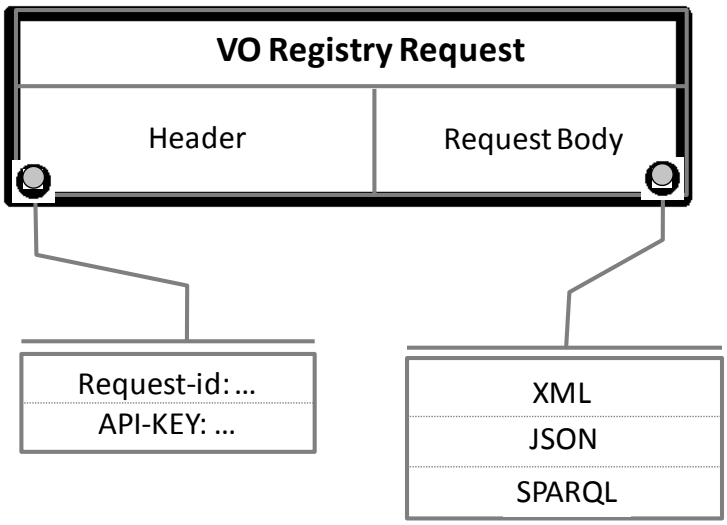

Figure 4: VO Registry Request Structure overview

Datatype	Description
Request Header(Request-id, API-KEY)	The Request head includes some data that are filled automatically by the system, such the Request id, Request Type, while it includes the API-KEY. The API-KEY is a string that corresponds to a unique identifier for the end-user entity which uses the VO Registry API.
Request Body	String that corresponds to the data of the VORegistryRequest payload. Depending on the situation, it may include XML, JSON, SPARQL data types.

Table 3: VO Registry Request - Description of contents

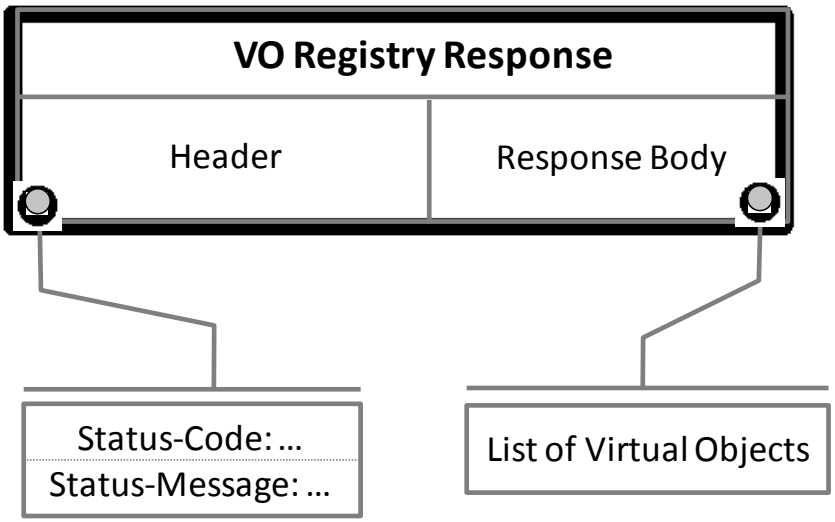

Figure 5: VO Registry Response Structure overview

Datatype	Description
Status-Code	Integer that presents the code of the response status, (e.g. 200).
Status-Message	String that presents the message of the response, (e.g. “Success”).
List of Virtual Objects	A List of URIs that corresponds to the VOs, which match the *VORegistryRequest*. It is null in case the status code is 404, namely in case there are no available VOs that match the request.

Table 4: VO Registry Response - Description of contents

3.4 VO Registry cURL API

3.4.1 API Overview

Since the VO Registry communication interface is based on the HTTP-REST, it has been designed and developed a cURL API for the interaction with the VO Registry, by aiming, in the same time, to enhancement of the interoperability aspects since the cURL functional aspects are supported by several programming languages such C++, php, etc. In order to use the cURL API, it is assumed that you have obtained an API-KEY, as well as that you have installed cURL (comes with Mac OS X and most Linux and BSD distributions).

For the support of the cURL API they have been developed the structure of the VO Registry Response and Request in form of XML and JSON. Thus, a cURL user can perform specific cURL commands either through its console or via appropriate code in any programming language, which supports cURL, The Table 5 and Table 6 below presents the cURL commands that should be executed in order to perform the corresponding interactions with the VO Registry.

Interaction Type	Data Format	cURL Command
Discovery Request	TEXT/XML APPLICATION/XML	curl -X POST --data "voregistryrequest=<XML_DATA>" http://<IP>:<PORT>/vo_registry/discovery.xml
	APPLICATION/JSON	curl -X POST --data "voregistryrequest=<JSON_DATA>" http://<IP>:<PORT>/vo_registry/discovery.json
Registration Request	TEXT/XML APPLICATION/XML	curl -X POST --data "voregistryrequest=<XML_DATA>" http://<IP>:<PORT>/vo_registry/registration.xml
	APPLICATION/JSON	curl -X POST --data "voregistryrequest=<JSON_DATA>" http://<IP>:<PORT>/vo_registry/registration.json
Update Request	TEXT/XML APPLICATION/XML	curl -X POST --data "voregistryrequest=<XML_DATA>" http://<IP>:<PORT>/vo_registry/update.xml
	APPLICATION/JSON	curl -X POST --data "voregistryrequest=<JSON_DATA>" http://<IP>:<PORT>/vo_registry/update.json
Delete Request	TEXT/XML APPLICATION/XML	curl -X POST --data "voregistryrequest=<XML_DATA>" http://<IP>:<PORT>/vo_registry/delete.xml
	APPLICATION/JSON	curl -X POST --data "voregistryrequest=<JSON_DATA>" http://<IP>:<PORT>/vo_registry/delete.json

Table 5: cURL Commands - Write Request data directly into the terminal

Interaction Type	Data Format	cURL Command
Discovery Request	TEXT/XML APPLICATION/XML	curl -X POST --data-urlencode voregistryrequest@filename.xml http://<IP>:<PORT>/vo_registry/discovery.xml
	APPLICATION/JSON	curl -X POST --data-urlencode voregistryrequest@filename.json http://<IP>:<PORT>/vo_registry/discovery.json
Registration Request	TEXT/XML APPLICATION/XML	curl -X POST --data-urlencode voregistryrequest@filename.xml http://<IP>:<PORT>/vo_registry/registration.xml
	APPLICATION/JSON	curl -X POST --data-urlencode voregistryrequest@filename.json http://<IP>:<PORT>/vo_registry/registration.json
Update Request	TEXT/XML APPLICATION/XML	curl -X POST --data-urlencode voregistryrequest@filename.xml http://<IP>:<PORT>/vo_registry/update.xml
	APPLICATION/JSON	curl -X POST --data-urlencode voregistryrequest@filename.json http://<IP>:<PORT>/vo_registry/update.json
Delete Request	TEXT/XML APPLICATION/XML	curl -X POST --data-urlencode voregistryrequest@filename.xml http://<IP>:<PORT>/vo_registry/delete.xml

	APPLICATION/JSON	curl -X POST --data-urlencode voregistryrequest@filename.json http://<IP>:<PORT>/vo_registry/delete.json

Table 6: cURL Commands - Read Request data from file

The **Table 7** presents samples of the VO Registry Requests structure, both in XML and JSON format. The user can use as the basis the presented samples in order to build their own VO Registry Requests, by performing any required interaction with the VO Registry.

VO REGISTRY REQUEST: REGISTRATION	
TEXT / XML |APPLICATION / XML	**APPLICATION / JSON**
<icore.voregistry.api.VORegistryRequest> <api__key>API-Key</api__key> <request__body> <![CDATA[<icore.voregistry.api.VirtualObject> ..... </icore.voregistry.api.VirtualObject>]]> </request__body> </VORegistryRequest>	**N/A**
VO REGISTRY REQUEST: DISCOVERY	
TEXT / XML |APPLICATION / XML	**APPLICATION / JSON**

<icore.voregistry.api.VORegistryRequest>
<api__key>API-KEY</api__key>
<request__body>
<![CDATA[
SELECT DISTINCT ?vo
WHERE {
$vo rdf:type icore:Virtual_Object.
}
]]>
</request__body>
</icore.voregistry.api.VORegistryRequest>
	{
	"api_key":"API-KEY",
	"request_body":
	"SELECT DISTINCT ?vo WHERE { $vo rdf:type icore:Virtual_Object . }"
	}

VO REGISTRY REQUEST: MODIFICATION (UPDATE/DELETE)	
TEXT / XML |APPLICATION / XML	**APPLICATION / JSON**

<icore.voregistry.api.VORegistryRequest>
<api__key>API-KEY==</api__key>
<request__body>
<![CDATA[
.....
]]>
</request__body>
</ icore.voregistry.api.VORegistryRequest>
	{
	"api_key":"API-KEY",
	"request_body":
	"DELETE { ?s ?p ?o . } WHERE { ?s ?p ?o . }"
	}

Table 7: VO Registry Request samples