The National Virtual Observatory Book
ASP Conference Series, Vol. 382, © 2008
M. J. Graham, M. J. Fitzpatrick, and T. A. McGlynn, eds.

Chapter 55: How to Use VOSpace

Matthew J. Graham

Introduction

In this tutorial, we will gain experience in setting up and using a VOSpace implementation. The software package contains both a VOSpace server and client and we shall be using both to retrieve data from other VO services and expose it to other users.

1. Implementation Details

The VOSpace implementation included in the software package consists of:

- a VOSpace web service - this can either be run as a standalone server (preferred method) or as a servlet running in Tomcat. The service is built on XFire 1.2RC1 Java SOAP framework (`http://xfire.codehaus.org`) and XmlBeans 2.1.0 (`http://xmlbeans.apache.org`).

- a VOSpace metadata store - this is a lightweight abstraction layer that sits on top of an existing database or other data store. Metastore implementations are provided for both the MySQL (`www.mysql.com`) and Virtuoso (`http://virtuoso.openlinksw.com/wiki/main/`) open source databases.

- an HTTP data server - the VOSpace implementation supports HTTP PUT and GET (v1.1) as transfer protocols. A simple HTTP server based on Apache HttpCore (`http://jakarta.apache.org/httpcomponents/httpcore/`) is provided: it interfaces with the VOSpace metadata store and employs one-time URLs.

- a VOSpace client - this is a simple command-line client to both the web service and the HTTP data server. It is built on Apache Axis 1.2.1 Final.

2. Installation

First we need to build the Java source code for the VOSpace server and client:

```
> source $NVOSS_HOME/bin/setup.csh
> cd $NVOSS_HOME/java/src/vospace
> ant jar
```

We also need to create a MySQL database to hold VOSpace metadata:

```
> mysql -u root -p<password> < schema.sql
```

3. Configuration

There are a number of properties that we need to set to configure the service. These are all in the file `$NVOSS_HOME/java/src/vospace/vospace.properties` and listed in Table 1.

Table 1. Configuration properties for VOSpace implementation

Parameter	Description	Suggested value
space.identifier	The IVOA identifier for the service	ivo://nvoss.org/vospace/name
space.baseuri	The root directory where the service will store data. Note this must be prefaced with the file:// scheme identifier	file:///tmp
server.space.port	This is the port that the service will run on	8008
server.meta.dbuid	The username used to access MySQL database	root
server.meta.dbpwd	The password used to access MySQL database	*password*
server.http.url	The external address of the HTTP data server	*Computer's IP address*
server.http.port	The port that the HTTP data server runs on.	7007
server.http.basedir	The root directory that the HTTP data server will use to store files: it should be the same as space.baseuri except without the file:// scheme identifier	/tmp

Note: if you are using a firewall, you will need to open the ports on which you are running the service and the HTTP data server.

4. Starting The Service

The service can be started by running the `vospaceserver` script:

```
> cd $NVOSS_HOME/java/src/vospace
> ./vospaceserver
DriverManager.getConnection("jdbc:mysql://localhost/vospace")
    trying
driver[className=com.mysql.jdbc.Driver,com.mysql.jdbc.Driver@ec6696]
getConnection returning
driver[className=com.mysql.jdbc.Driver,com.mysql.jdbc.Driver@ec6696]
```

```
2006-09-10 22:56:01,524 [Thread-0] INFO
edu.caltech.nvo.vospace.server.HttpDataServer - Accepting connections
on port 7007
2006-09-10 22:56:03,352 [main] INFO  org.mortbay.http.HttpServer -
Version Jetty/5.1.3
2006-09-10 22:56:03,362 [main] INFO  org.mortbay.util.Container -
Started org.mortbay.jetty.servlet.ServletHandler@8efa2f
2006-09-10 22:56:03,363 [main] INFO  org.mortbay.util.Container -
Started ServletHttpContext[/,/]
2006-09-10 22:56:03,369 [main] INFO  org.mortbay.http.SocketListener
- Started SocketListener on 0.0.0.0:8080
2006-09-10 22:56:03,370 [main] INFO  org.mortbay.util.Container -
Started org.mortbay.jetty.Server@a3da69
```

You will probably want to run this in a separate window.

5. Using The Client

The client can be run using the `vospaceclient` script: the syntax is:

```
> cd $NVOSS_HOME/java/src/vospace
> ./vospaceclient <method> <service endpoint>
```

where the `<service endpoint>` is the endpoint URL for the VOSpace service you want to call. For the VOSpace instance using this implementation, this will be:

```
http://localhost:8008/VOSpaceServiceImpl
```

Table 2. Methods available from vospaceclient

Method	Description
getviews	Gets a list of the available import and export data formats for the space
getprotocols	Gets a list of supported transfer protocols for the space
getproperties	Gets a list of the properties for the space
createnode	Creates a new node in the space
deletenode	Deletes a node from the space
movenode	Moves a node within the space
copynode	Copies a node within the space
Getnode	Gets the details for a specific node
Setnode	Set the property values for a specific node
listnodes	Lists the nodes in the space
pushtospace	Gets a URL to send data to the space
pulltospace	Imports data into the space
pullfromspace	Gets a URL that data can be read from
pushfromspace	Get the space to send data to a remote location
httpput	Performs an HTTP PUT operation
httpget	Performs an HTTP GET operation

The available methods are listed in Table 2. Most of these methods will prompt you for further information and are described in more detail in the VOSpace chapter (Chapter 40).

It is also possible to use the `vospace.jar` file that is built when the client source code is compiled (see Section 2) to add VOSpace client functionality to your own code. You will need to import the following packages:

```
import ivoa.net.www.xml.VOSpaceContract_v1_0rc6.*;
import ivoa.net.www.xml.VOSpaceTypes_v1_0rc6.*;
import ivoa.net.www.xml.VOSpaceTypes_v1_0rc6.holders.*;
```

and make sure that `vospace.jar` is in our `CLASSPATH` when we compile and execute our code. Note that this file contains code auto-generated from the VOSpace WSDL using Apache Axis 1.2.1 (see Chapter 59) and may not work if used with other versions of Axis.

6. Example: Saving Query Results Into VOSpace

In this first example, we will save the results of a cone search query into our local VOSpace service. Note that in this and the next example, user input is denoted in the code blocks with bold face text.

The first thing we want to do is to find a cone search service that we can use so we want to go to a registry and get a list of cone search services: Using the STScI registry, select the 'Contents' tab and then the 'CONE' link in the ResourceType column of the table. If you are using CARNIVORE, select the 'Summary' tab and then 'Browse' button in front of the Cone Search type. Once you have your list, pick a cone search and look at the XML record for the service - the value we are after is the `<accessURL>`.

We're going to use the GALEX cone search at `http://galex.stsci.edu/gxWS/ConeSearch/gxConeSearch.aspx?` in this example. Now remember that the cone search call also requires the parameters: `ra`, `dec` and `sr`. Our complete URL will then be:

```
http://galex.stsci.edu/gxWS/ConeSearch/gxConeSearch.aspx?ra=120&dec=2
0&sr=0.5
```

Cone search services return VOTables so we want to make sure that our VOSpace service supports the VOTable data format (using the `getviews` method):

```
> ./vospaceclient getviews http://localhost:8008/VOSpaceServiceImpl
Get Views
=========
The available import data formats for the space are:
ivo://net.ivoa.vospace/views/image/png-1.0
ivo://net.ivoa.vospace/views/image/jpeg-1.0
ivo://net.ivoa.vospace/views/image/fits-image
ivo://net.ivoa.vospace/views/tabular/votable-1.1
```

```
ivo://net.ivoa.vospace/views/tabular/fits-table

The available export data formats for the space are:
ivo://net.ivoa.vospace/views/image/png-1.0
ivo://net.ivoa.vospace/views/image/jpeg-1.0
ivo://net.ivoa.vospace/views/image/fits-image
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
```

Note the URI for VOTable: `ivo://net.ivoa.vospace/views/tabular/votable-1.1` is available as both an import and export data format.

We are also going to be retrieving the cone search results using an HTTP GET call, so we want to check that our VOSpace service supports this protocol (using the `getprotocols` method):

```
> ./vospaceclient getprotocols
http://localhost:8008/VOSpaceServiceImpl
Get Protocols
=============
The transfer protocols for which the space can act as a client are:
ivo://net.ivoa.vospace/protocol/http-get
ivo://net.ivoa.vospace/protocol/http-put

The transfer protocols for which the space can act as a server are:
ivo://net.ivoa.vospace/protocol/http-get
ivo://net.ivoa.vospace/protocol/http-put
```

The URI for HTTP GET is: `ivo://net.ivoa.vospace/protocol/http-get`.

We're now ready to tell the VOSpace service to call this cone search service and pull the results over to it (using the `pulltospace` method):

```
>./vospaceclient pulltospace http://localhost:8008/VOSpaceServiceImpl
Pull data to space
==================
Define the node into which data is to be imported:
What type of node (plain, data, unstructured, structured):
```

I'm going to use a structured node:

```
What type of node (plain, data, unstructured, structured): structured
What is the identifier for the node (hitting return will cause the
space to generate an identifier):
```

Now we need to fill in the identifier for this node: remember that this needs to conform to the VOSpace identifier syntax (*vos://[service]/[name]*). When you configured your VOSpace service, you specified an IVOA identifier for your service - the value of the `space.identifier` property. If you used the suggested value of `ivo://nvoss.org/vospace/name` then the *[service]* part of the VOSpace identifier

will be: `nvoss.org!vospace!name`. The *[name]* part can be whatever you want, but we're going to use `votable1`:

```
What is the identifier for the node (hitting return will cause the
space to generate an identifier):vos://nvoss.org!vospace!mjg/votable1
Now for the node properties:
Identifying URI for property (blank means no more properties):
```

We're not going to define any properties on this node so I just hit Return

```
Now specify the transfer details:
What is the identifier for the data format:
```

The data format is VOTable so we just put in the URI we noted above:

```
What is the identifier for the data format:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
Now for the protocols:
Identifying URI for protocol (blank means no more protocols):
```

and the protocol we are using is HTTP GET:

```
Identifying URI for protocol (blank means no more protocols):
ivo://net.ivoa.vospace/protocol/http-get
What is the endpoint for this protocol:
```

and this is where we put the complete URL for the cone search that we constructed above:

```
What is the endpoint for this protocol:
http://galex.stsci.edu/gxWS/ConeSearch/gxConeSearch.aspx?ra=120&dec=2
0&sr=0.5
Identifying URI for protocol (blank means no more protocols):
```

We're not going to specify any alternate transfer protocols because there are none for a cone search so we just hit RETURN. The output we see next is:

```
The node into which data was imported in now defined as:
Identifier: vos://nvoss.org!vospace!mjg/votable1
Type: StructuredDataNodeType
Properties:
No properties are defined.
Accepted data formats:
ivo://net.ivoa.vospace/views/image/png-1.0
ivo://net.ivoa.vospace/views/image/jpeg-1.0
ivo://net.ivoa.vospace/views/image/fits-image
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
```

```
Provided data formats:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
```

Well, this seems to have worked OK but let's just check. In the `vospace.properties` file, we specified the root directory where the HTTP data server will store data in the `space.http.baseuri` parameter (the suggested value is `/tmp`) so let's just have a look in this directory:

```
> ls /tmp
nvoss.org_vospace_mjg_votable1
```

and we can check that it looks reasonable as well:

```
> head /tmp/nvoss.org_vospace_mjg_votable1
<?xml version="1.0" encoding="utf-8"?>
<VOTABLE xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://galex.stsci.edu">
  <DESCRIPTION> Results from MAST/GALEX Service </DESCRIPTION>
  <DEFINITIONS>
    <COOSYS ID="J2000" equinox="2000." epoch="2000." system="eq_FK5"
/>
  </DEFINITIONS>
  <INFO name="SUCCESS" value="Parameters validated" />
  <RESOURCE type="results">
    <DESCRIPTION>GALEX Conesearch Service Results</DESCRIPTION>
    <INFO name="QUERY_STATUS" value="OK" />
```

Now we'll have a look at how we could do this example programmatically instead using the `vospace.jar` file. We're just going to give the main details and miss out things like exception handling (`try…catch` blocks) which we leave as an exercise for the developer (or alternatively have a look at `$NVOSS_HOME/java/src/vospace/src/java/VOSpaceClient.java`). `vospace.jar` contains client stubs (see Chapter 59) for the VOSpace service so we first need to instantiate a service stub:

```
VOSpaceServiceLocator locator = new VOSpaceServiceLocator();
locator.setVOSpacePortEndpointAddress("endpoint");
service = locator.getVOSpacePort();
```

where `endpoint` is the endpoint URL for the service. We have to create and fill in the details for the node that we will be saving our results to:

```
NodeType node = new StructuredDataNodeType();
node.setUri(new URI("identifier"));
// node.setProperties(new PropertyListType());
```

where `identifier` is the VOSpace identifier for our node. If we wanted to attach any properties to our data node then we would need to create and populate a `PropertyListType` object. We also have to specify the data format (view) and the transfer protocols that we want to try:

```
ViewType view = new ViewType();
view.setUri(new URI("identifier"));
ProtocolType[] protocols = new ProtocolType[2];
protocols[0].setUri(new URI("identifier"));
protocols[1].setUri(new URI("identifier"));
TransferType transfer = new TransferType(view, protocols);
```

where the various `identifier`s take their appropriate values (see above). If we want the VOSpace service to call the cone search service directly then we need to set the endpoint URL for the HTTP GET protocol that will be used to retrieve the query results:

```
protocols[0].setEndpoint(new URI("cone search query URI"));
```

and we can then just call the service:

```
NodeTypeHolder nodeHolder = new NodeTypeHolder(node);
TransferTypeHolder transferHolder = new TransferTypeHolder(transfer);
Service.pullToVoSpace(nodeHolder, transferHolder);
```

The updated information for the node into which we transferred the results is accessible as `nodeHolder.value`.

If we are adding this functionality to existing code where we already have the cone search results, e.g. modifying the cone search client in Chapter 46 then we will actually be transferring the data to VOSpace ourselves and not getting the service to do it on our behalf. In this case, an appropriate protocol would be HTTP PUT and we would need to make sure that it was one of the protocols set.

```
NodeTypeHolder nodeHolder = new NodeTypeHolder(node);
TransferTypeHolder transferHolder = new TransferTypeHolder(transfer);
service.pushToVOSpace(nodeHolder, transferHolder);
```

The updated information for the node is again accessible as `nodeHolder.value` and the details of the transfer are in `transferHolder.value`. We would then transfer the data with the agreed protocol with something like:

```
HttpClient httpClient = new HttpClient();
PutMethod put = new PutMethod(transferHolder.value.getEndpoint());
put.setRequestBody(new FileInputStream("file"));
httpClient.executeMethod(put);
```

where "`file`" is where we have temporarily stored the cone search results.

7. Example: Third-party Transfers Between Friends

In this second example, we're going to transfer some data from one VOSpace to another, so let's imagine that we have a group of three friends: Ada, Byron, and Charles. Each is going to transfer data from the VOSpace of the first of the other two in the group to the VOSpace of the second of the other two in the group: so Ada is going to transfer data from Byron's VOSpace to Charles', Byron will transfer data from Charles' VOSpace to Ada's and Charles will move data from Ada's VOSpace to Byron's.

Firstly you're going to have to note the IP addresses of each of the VOSpaces. For our example, see Table 3.

Table 3. IP addresses of people in this example

Person	IP address
Ada	192.168.1.201
Byron	192.168.1.202
Charles	192.168.1.203

Since the only data in any of our VOSpaces is the result of a cone search (from the first example), we already know that the data format is going to be VOTable so the URI is: `ivo://net.ivoa.vospace/views/tabular/votable-1.1`.

We're going to get one VOSpace to push data to another so the transfer protocol is HTTP PUT. If we were really dealing with a set of different VOSpace implementations, we would check first of all that each of the two VOSpaces could deal with HTTP PUT (although the VOSpace implementation will do this automatically as runtime). However, since we are all using the same implementation, we just need to check the one that we already did in the previous example.

Finally, we need to know what the identifier of the data we are transferring is (see Table 4):

Table 4. Identifier examples.

Person	Source identifier	Destination identifier
Ada	vos://nvoss.org!vospace!Byron/votable1	vos://nvoss.org!vospace!Charles/votable2
Byron	vos://nvoss.org!vospace!Charles/votable1	vos://nvoss.org!vospace!Ada/votable2
Charles	vos://nvoss.org!vospace!Ada/votable1	vos://nvoss.org!vospace!Byron/votable2

Of course, in real life you would have done a listing of the VOSpace from which you are transferring data.

We're now going to follow what Ada does - firstly she needs to talk to Charles' VOSpace to tell it that data is going to be pushed to it, and to get an HTTP endpoint to receive it. We use the `pushtospace` method to notify Charles' VOSpace, and read the endpoint from the transfer details returned by the task:

```
> ./vospaceclient pushtospace
http://192.168.1.202:8008/VOSpaceServiceImpl
Push data to space
==================
Define the node into which data is to be imported:
What type of node (plain, data, unstructured, structured): structured
What is the identifier for the node (hitting return will cause the
space to generate an identifier):
vos://nvoss.org!vospace!Charles/vospace2
Now for the node properties:
Identifying URI for property (blank means no more properties):
Now specify the transfer details:
What is the identifier for the data format:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
Now for the protocols:
Identifying URI for protocol (blank means no more protocols):
ivo://net.ivoa.vospace/protocol/http-put
Identifying URI for protocol (blank means no more protocols):
The node into which data is to be imported is now defined as:
Identifier: vos://nvoss.org!vospace!Charles/vospace2
Type: StructuredDataNodeType
Properties:
No properties are defined.
Accepted data formats:
ivo://net.ivoa.vospace/views/image/png-1.0
ivo://net.ivoa.vospace/views/image/jpeg-1.0
ivo://net.ivoa.vospace/views/image/fits-image
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
Provided data formats:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table

The operational details of the transfer are:
Format: ivo://net.ivoa.vospace/views/tabular/votable-1.1
Protocol: ivo://net.ivoa.vospace/protocol/http-put
Endpoint: http://192.168.1.202:7007/ece6fa21-ec58-4ef2-9e37-
dfb778de4e74
```

Now that Ada has the transfer details, she just needs to tell Byron's VOSpace to push the data from itself to the HTTP PUT endpoint that Charles' VOSpace provided her with:

```
> ./vospaceclient pushfromspace
http://192.168.1.201:8008/VOSpaceServiceImpl
Push data from space
====================
```

```
What is the identifier of the node from which data is to be exported:
vos://nvoss.org!vospace!Byron/vospace1
Now specify the transfer details:
What is the identifier for the data format:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
Now for the protocols:
Identifying URI for protocol (blank means no more protocols):
ivo://net.ivoa.vospace/protocol/http-put
What is the endpoint for this protocol:
http://192.168.1.202:7007/ece6fa21-ec58-4ef2-9e37-dfb778de4e74
Identifying URI for protocol (blank means no more protocols):
```

Ada can now do a listing of Charles' VOSpace to see if the data is there:

```
> ./vospaceclient listnodes
http://192.168.1.202:8008/VOSpaceServiceImpl
List Nodes
==========
What level of detail do you want in the listing (min|properties|max):
max
Identifier: vos://nvoss.org!vospace!Charles/votable2
Type: StructuredDataNodeType
Properties:
No properties are defined.
Accepted data formats:
ivo://net.ivoa.vospace/views/image/png-1.0
ivo://net.ivoa.vospace/views/image/jpeg-1.0
ivo://net.ivoa.vospace/views/image/fits-image
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
Provided data formats:
ivo://net.ivoa.vospace/views/tabular/votable-1.1
ivo://net.ivoa.vospace/views/tabular/fits-table
```

and/or she can ask Charles to check that his data is in the VOSpace and looks OK.

8. Example: Retrieving Results From A Saved Query

In this final example, we are going to extract some data (an accessReference) from the results of a SIAP query that we saved in our VOSpace previously (using the same methodology as in the first example). Note that the DAL query (cone search and SIAP) in these examples is being done as part of the VOSpace method.

We are going to consider how to do this programmatically, i.e. using vospace.jar. The first thing that we need to do is specify the data format (view) and the transfer protocols that we want to use:

```
ViewType view = new ViewType();
view.setUri(new URI("ivo://net.ivoa.vospace/views/tabular/votable-
1.1"));
ProtocolType[] protocols = new ProtocolType[1];
protocols[0].setUri(new URI("ivo://net.ivoa.vospace/protocol/http-
get"));
TransferType transfer = new TransferType(view, protocols);
TransferTypeHolder transferHolder = new TransferTypeHolder(transfer);
```

and then we can just request the data object we are interested in using its VOSpace identifier:

```
service.pullFromVoSpace(new URI("vos://nvoss.org!vospace!mjg/siap1"),
                            transferHolder);
```

The details of the transfer are again in `transferHolder.value`:

```
URI endpoint = transferHolder.value.getProtocol().getEndpoint();
InputStream is = new URL(URI.toString()).openStream();
VOTWrap.VOTable vot = VOTWrap.createVOTable(is);
```

We can now use our regular suite of programmatic tools (see Chapter 34) to extract the information we want from the VOTable.

Useful Links

The Sloan Digital Sky Survey (SDSS). Aug 2006; http://skyserver.sdss.org/ [Accessed 31 Oct 2006].
Galaxy Evolution Explorer (GALEX). March 2006; http://galex.stsci.edu/ [Accessed 31 Oct 2006].
SuperCOSMOS Science Archive (SSA). Jan 2006; http://surveys.roe.ac.uk/ [Accessed 31 Oct 2006].