



International

Virtual

Observatory

Alliance

VOSpace service specification

Version 1.02

IVOA Recommendation 2008 January 24

This version:

<http://www.ivoa.net/Documents/REC/GWS/VOSpace-20080124.pdf>

Latest version:

<http://www.ivoa.net/Documents/latest/VOSpace.html>

Previous version(s):

PR 1.02 <http://www.ivoa.net/Documents/PR/GWS/VOSpace-20070907.pdf>

PR 1.01 <http://www.ivoa.net/Documents/PR/GWS/VOSpace-20070723.pdf>

PR 1.0 <http://www.ivoa.net/Documents/PR/GWS/VOSpace-20070626.pdf>

WD 1.0 <http://www.ivoa.net/Documents/WD/GWS/VOSpace-20070304.pdf>

0.21 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/vospace-0.21.doc>

0.20 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/vospace-0.20.doc>

0.19 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/vospace-0.19.doc>

0.18 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/VOStore0.18.pdf>

0.17 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/VOStore0.17.pdf>

0.15 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/VOStore0.14.pdf>

0.13 <http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/VOStore0.13.pdf>

Author(s):

Matthew Graham (editor for this version)

Paul Harrison

Dave Morris

Guy Rixon

Abstract

VOSpace is a SOAP interface for access to data stores.

This version applies the VOSpace concept to flat, unconnected data spaces.

Future versions of the specification will add extensions to support a hierarchical structure and links between the individual space services.

Status of this document

This document has been produced by the IVOA Grid and Web Services Working Group.

It has been reviewed by IVOA Members and other interested parties, and has been endorsed by the IVOA Executive Committee as an IVOA Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. IVOA's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability inside the Astronomical Community.

A list of current IVOA Recommendations and other technical documents can be found at <http://www.ivoa.net/Documents/>.

Acknowledgements

This document derives from discussions among the Grid and Web Services working group of the IVOA. It is particularly informed by prototypes built by Matthew Graham (Caltech/NVO), Paul Harrison (ESO/EuroVO) and David Morris (Cambridge, Astrogrid).

This document has been developed with support from the National Science Foundation's Information Technology Research Program under Cooperative Agreement AST0122449 with the John Hopkins University, from the UK Particle Physics and Astronomy Research Council (PPARC), and from the European Commission's Sixth Framework Program via the Optical Infrared Coordination Network (OPTICON).

Conformance related definitions

The words "MUST", "SHALL", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in IETF standard, RFC 2119 [RFC 2119].

The **Virtual Observatory (VO)** is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The **International Virtual Observatory Alliance (IVOA)** is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications. The International Virtual Observatory (IVO) application is an application that takes advantage of IVOA standards and infrastructure to provide some VO service.

Contents

| | |
|--|---|
| Abstract | 1 |
| Status of this document | 2 |
| Acknowledgements | 2 |
| Conformance related definitions | 2 |
| Contents | 2 |
| 1 Introduction | 5 |
| 1.1 Typical use of a VOSpace service | 5 |
| 1.2 Document roadmap | 8 |

| | | |
|---------|--|----|
| 2 | VOspace identifiers | 9 |
| 3 | VOspace data model | 11 |
| 3.1 | Nodes and node types | 12 |
| 3.2 | Properties | 13 |
| 3.2.1 | Property values | 13 |
| 3.2.2 | Property identifiers | 13 |
| 3.2.3 | Property descriptions | 14 |
| 3.2.3.1 | UI display name | 14 |
| 3.2.3.2 | Property editors | 14 |
| 3.2.4 | Standard properties | 15 |
| 3.3 | Views | 15 |
| 3.3.1 | Example use cases | 15 |
| | Simple file store | 15 |
| 3.3.1.1 | Database store | 16 |
| 3.3.2 | View identifiers | 17 |
| 3.3.3 | View descriptions | 17 |
| 3.3.3.1 | UI display name | 18 |
| 3.3.3.2 | MIME types | 18 |
| 3.4 | Protocols | 18 |
| 3.4.1 | Protocol identifiers | 18 |
| 3.4.2 | Protocol descriptions | 19 |
| 3.4.2.1 | UI display name | 19 |
| 3.4.3 | Standard protocols | 19 |
| 3.4.4 | Custom protocols | 19 |
| 3.4.4.1 | SRB gateway | 20 |
| 3.4.4.2 | Local NFS transfers | 20 |
| 3.5 | Transfers | 21 |
| 3.5.1 | Synchronous transfers | 21 |
| 3.5.2 | Asynchronous transfers | 22 |
| 4 | Access control | 23 |
| 5 | Web service operations | 23 |
| 5.1 | Service metadata | 24 |
| 5.1.1 | getProtocols | 24 |
| 5.1.1.1 | Parameters | 24 |
| 5.1.1.2 | Returns | 24 |
| 5.1.1.3 | Faults | 24 |
| 5.1.2 | getViews | 24 |
| 5.1.2.1 | Parameters | 24 |
| 5.1.2.2 | Returns | 24 |
| 5.1.2.3 | Faults | 25 |
| 5.1.3 | getProperties | 25 |
| 5.1.3.1 | Parameters | 25 |
| 5.1.3.2 | Returns | 25 |
| 5.1.3.3 | Faults | 25 |
| 5.2 | Creating and manipulating data nodes | 25 |
| 5.2.1 | createNode | 25 |
| 5.2.1.1 | Parameters | 25 |
| 5.2.1.2 | Returns | 26 |
| 5.2.1.3 | Faults | 26 |
| 5.2.2 | deleteNode | 26 |

| | | |
|---------|------------------------------------|----|
| 5.2.2.1 | Parameters | 26 |
| 5.2.2.2 | Returns | 26 |
| 5.2.2.3 | Faults | 26 |
| 5.2.3 | listNodes | 26 |
| 5.2.3.1 | Parameters | 27 |
| 5.2.3.2 | Returns | 28 |
| 5.2.3.3 | Faults | 28 |
| 5.2.4 | moveNode | 28 |
| 5.2.4.1 | Parameters | 29 |
| 5.2.4.2 | Returns | 29 |
| 5.2.4.3 | Faults | 29 |
| 5.2.5 | copyNode | 29 |
| 5.2.5.1 | Parameters | 29 |
| 5.2.5.2 | Returns | 30 |
| 5.2.5.3 | Faults | 30 |
| 5.3 | Accessing metadata | 30 |
| 5.3.1 | getNode | 30 |
| 5.3.1.1 | Parameters | 30 |
| 5.3.1.2 | Returns | 30 |
| 5.3.1.3 | Faults | 30 |
| 5.3.2 | setNode | 30 |
| 5.3.2.1 | Parameters | 31 |
| 5.3.2.2 | Returns | 31 |
| 5.3.2.3 | Faults | 31 |
| 5.4 | Transferring data | 31 |
| 5.4.1 | pushToVoSpace | 31 |
| 5.4.1.1 | Parameters | 31 |
| 5.4.1.2 | Returns | 32 |
| 5.4.1.3 | Faults | 32 |
| 5.4.2 | pullToVoSpace | 32 |
| 5.4.2.1 | Parameters | 33 |
| 5.4.2.2 | Returns | 33 |
| 5.4.2.3 | Faults | 33 |
| 5.4.2.4 | Notes | 33 |
| 5.4.3 | pullFromVoSpace | 34 |
| 5.4.3.1 | Parameters | 34 |
| 5.4.3.2 | Returns | 34 |
| 5.4.3.3 | Faults | 34 |
| 5.4.3.4 | Notes | 34 |
| 5.4.4 | pushFromVoSpace | 35 |
| 5.4.4.1 | Parameters | 35 |
| 5.4.4.2 | Returns | 35 |
| 5.4.4.3 | Faults | 35 |
| 5.4.4.4 | Notes | 35 |
| 5.5 | Fault arguments | 36 |
| 5.5.1 | InternalFault | 36 |
| 5.5.2 | PermissionDenied | 36 |
| 5.5.3 | InvalidURI | 36 |
| 5.5.4 | NodeNotFound | 36 |
| 5.5.5 | DuplicateNode | 36 |

| | | |
|--------|--|----|
| 5.5.6 | InvalidToken | 36 |
| 5.5.7 | InvalidArgument | 36 |
| 5.5.8 | TypeNotSupported | 36 |
| 5.5.9 | ViewNotSupported | 36 |
| 5.5.10 | InvalidData | 36 |
| 6 | Appendix: Machine readable definitions | 37 |
| 6.1 | WSDL | 37 |
| 6.2 | Message schema | 63 |
| 7 | References | 88 |

1 Introduction

VOspace is an interface standard for data stores. It specifies how VO agents and applications can use network attached data stores to persist and exchange data in a standard way.

A VOspace web service is an access point for a distributed storage network. Through this access point, a client can :

- add or delete data objects
- manipulate metadata for the data objects
- obtain URIs through which the content of the data objects can be accessed

VOspace does not define how the data is stored or transferred, only the control messages to gain access. Thus, the VOspace interface can readily be added to an existing storage system.

When we speak of “a VOspace”, we mean the arrangement of data accessible through one particular VOspace service.

Each data object within a VOspace service is represented as a node. A useful analogy to have in mind when reading this document is that a node is equivalent to a file.

Nodes in VOspace have unique identifiers expressed as URIs in the 'vos://' scheme, as defined below.

In this version of the standard, each VOspace service provides a single, flat set of data objects, similar to a service described by the earlier VOStore standard; this version of the VOspace specification supersedes VOStore.

Future versions of the VOspace specification may provide support for a hierarchical arrangement of objects within a space, and may provide support for VOspace services to be linked such that a client can navigate them as one global space.

Services implementing the current version of the specification can be linked in as leaf nodes of this global space without needing to change.

1.1 Typical use of a VOspace service

A typical use case for VOspace is uploading a local data file to a remote VOspace service. The user will specify the name for the data file in the VOspace (its node identifier), and any metadata (its properties) that they want to associate with it, e.g. MIME type. They will then describe the data format (the view) they want to use in uploading the file, e.g. VOTable, and the transport protocol (the protocol) that they want to employ to upload the

file, e.g. HTTP PUT. This will result in a SOAP request to the service resembling this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <soapenv:Body>
      <PushToVoSpace xmlns="http://www.net.ivoa/xml/VOspaceContract-v1.0"
        xmlns:vost="http://www.net.ivoa/xml/VOspaceTypes-v1.0">
        <vost:destination uri="vos://nvo.caltech!vospace/mytable1"
          xsi:type="vost:UnstructuredDataNodeType">

          <vost:properties>
            <vost:property uri="ivo://ivoa.net/vospace/core#mimetype"
              xsi:type="vost:PropertyType">text/xml</vost:property>
          </vost:properties>
        </vost:destination>
        <vost:transfer>
          <vost:view uri="ivo://ivoa.net/vospace/core#votable"/>
          <vost:protocol uri="ivo://ivoa.net/vospace/core#http-put"/>
        </vost:transfer>
      </PushToVoSpace>
    </soapenv:Body>
  </soapenv:Envelope>
```

The service will reply with the representation of the data file in the VOspace (a description of the node) and the details of the data transfer, i.e. the URL that the user will PUT the data file to. This will involve a SOAP response similar to:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <soapenv:Body>
      <PushToVoSpaceResponse xmlns="http://www.net.ivoa/xml/VOspaceContract-v1.0"
        xmlns:vost="http://www.net.ivoa/xml/VOspaceTypes-v1.0">
        <vost:destination uri="vos://nvo.caltech!vospace/mytable1"
          xsi:type="vost:UnstructuredDataNodeType">

          <vost:properties>
            <vost:property uri="ivo://ivoa.net/vospace/core#mimetype"
              xsi:type="vost:PropertyType">text/xml</vost:property>
          </vost:properties>
        </vost:destination>
        <vost:transfer>
          <vost:view uri="ivo://ivoa.net/vospace/core#votable"/>
          <vost:protocol uri="ivo://ivoa.net/vospace/core#http-put">
            <vost:endpoint>http://nvo.caltech.edu:
7777/aabcf5-348874-9873ca-9a9ab4</vost:endpoint>
          <vost:protocol>
```

```

    </vost:transfer>
  </PushToVoSpaceResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The user will then use a regular HTTP client to transfer (PUT) the local file to the specified endpoint. This illustrates an important point about VOspace – it is only concerned with the management of data storage and transfer. A client negotiates the details of a data transfer with a VOspace service but the actual transfer of bytes across a network is handled by other tools.

Similarly, when a user wants to retrieve a data file from a VOspace service, they will specify the identifier for the data file in the VOspace, the data format (view) they want to use in downloading the file, e.g. VOTable, and the transport protocol (the protocol) that they want to employ to download the file, e.g. HTTP GET. This will result in a SOAP request to the service resembling this:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <soapenv:Body>
      <PullFromVoSpace xmlns="http://www.net.ivoa/xml/VOspaceContract-v1.0"
        xmlns:vost="http://www.net.ivoa/xml/VOspaceTypes-v1.0">
        <vost:source>vos://nvo.caltech!vospace/mytable1</vost:source>
        <vost:transfer>
          <vost:view uri="ivo://ivoa.net/vospace/core#votable"/>
          <vost:protocol uri="ivo://ivoa.net/vospace/core#http-get"/>
        </vost:transfer>
      </PullFromVoSpace>
    </soapenv:Body>
  </soapenv:Header>
</soapenv:Envelope>

```

The service will reply with a SOAP response similar to:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <soapenv:Body>
      <PullFromVoSpaceResponse xmlns="http://www.net.ivoa/xml/VOspaceContract-
v1.0"
        xmlns:vost="http://www.net.ivoa/xml/VOspaceTypes-v1.0">
        <vost:transfer>
          <vost:view uri="ivo://ivoa.net/vospace/core#votable"/>
          <vost:protocol uri="ivo://ivoa.net/vospace/core#http-get">
            <vost:endpoint>http://nvo.caltech.edu:7777/79f45-3ab0-4de2-bd6c-
ff016082fd</vost:endpoint>
          <vost:protocol>
        </vost:transfer>
      </PullFromVoSpaceResponse>
    </soapenv:Body>
  </soapenv:Header>
</soapenv:Envelope>

```

```
</PullFromVoSpaceResponse>
</soapenv:Body>
</soapenv:Envelope>
```

The user can then download the data file by pointing an HTTP client (e.g. web browser) at the specified endpoint.

1.2 Document roadmap

The rest of this document is structured as follows:

In section 2, we specify the URI syntax for identifying data objects (nodes) in VOSpace.

In section 3, we present the data model that underpins the VOSpace architecture. This consists of a number of data structures, which have XML representations that are used across the wire in SOAP messages to and from a VOSpace service. These structures represent the data objects themselves (nodes), metadata that can be associated with a data object (properties), the data format used when transferring data objects across the wire (views), the transport protocol employed in a data transfer (protocols) and the data transfer itself (transfers).

In section 4, we outline how access control policies are currently handled in VOSpace.

In section 5, we detail the operations that the VOSpace interface supports and exposes through its WSDL. These handle access to service-level metadata, the creation and manipulation of nodes within the VOSpace, access to node metadata (properties) and data transfer to and from the VOSpace.

Finally in section 6, we include the standard WSDL for VOSpace 1.0 and its message schema for reference.

In addition to this specification, there will be an ancillary document (in production) giving more worked examples of using the interface and also giving usage policy, such as a list of identifiers for core metadata (views, properties and transport protocols).

2 VOSpace identifiers

The identifier for a node in VOSpace shall be a URI with the scheme `vos://`.

Such a URI shall have the following parts with the meanings and encoding rules defined in RFC2396 [2].

- scheme
- naming authority
- path
- (optional) query
- (optional) fragment identifier

The naming authority for a VOSpace node shall be the VOSpace service through which the node was created. The authority part of the URI shall be constructed from the IVO registry identifier [2] for that service by deleting the `ivo://` prefix and changing all forward-slash characters ('/') in the resource key to exclamation marks (!).

This is an example of a possible VOSpace identifier.

```
vos://org.astrogrid.cam!vospace!container-6/siap-out-1.vot?foo=bar#baz
```

- The URI scheme is `vos://`

Using a separate URI scheme for VOSpace identifiers enables clients to distinguish between IVO registry identifiers and VOSpace identifiers.

- `org.astrogrid.cam!vospace!container-6`

is the authority part of the URI, corresponding to the IVO registry identifier

- `ivo://org.astrogrid.cam/vospace/container-6`

This is the IVO registry identifier of the VOSpace service that contains the node.

- `/siap-out-1.vot` is the URI path

Slashes in the path imply a hierarchical arrangement of data, as is normal with URIs. Since the current version of the specification does not support data hierarchies, an identifier for a node in a current service must have one slash at the start of the path and no other slashes.

- `?foo=bar` is a query string and thus is something to which the VOSpace service is supposed to respond

No queries of this nature are defined in the current version of the specification, but the query string system is reserved for use in later versions of the VOSpace specification.

- `#baz` is a fragment identifier. Its meaning attaches to the data returned from the VOSpace node, not to the node itself.

The fragment identifier should be interpreted by the client, not by the VOSpace service; the service shall ignore any fragment identifiers.

A VOSpace identifier is globally unique, and identifies one specific node in a specific VOSpace service.

A client should use the following procedure to resolve access to a VOSpace node from a VOSpace identifier:

- Extract the authority part of the VOSpace URI
- Convert the authority back to the IVO registry identifier of the VOSpace service by changing any '!' characters to '/' and adding the `ivo://` prefix
- Resolve the IVO registry identifier to an endpoint for the VOSpace service using the IVO resource registry
- Access the node via the endpoint using one of the web service methods defined in this standard

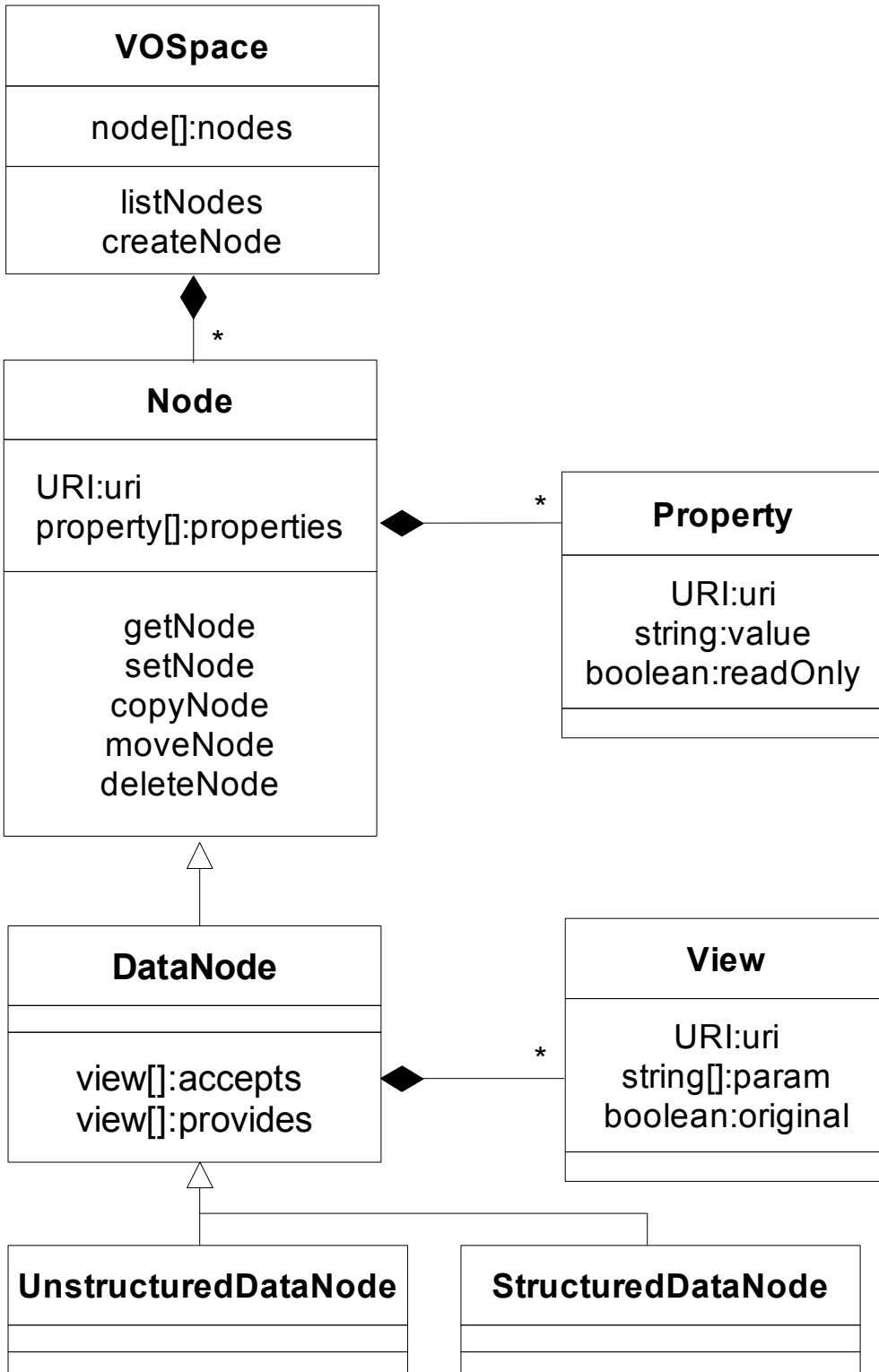
Given the example identifier

```
vos://org.astrogrid.cam!vospace!container-6/siap-out-1.vot?foo=bar#baz
```

processing the URI to resolve the VOSpace service would involve :

- Extract the authority part of the VOSpace URI
 - `org.astrogrid.cam!vospace!container-6`
- Convert the authority back to the IVO registry identifier of the VOSpace service by changing any '!' characters to '/'
 - `org.astrogrid.cam/vospace/container-6`
- Adding the `ivo://` prefix
 - `ivo://org.astrogrid.cam/vospace/container-6`
- Using the IVO registry to resolve the VOSpace service endpoint from the IVO identifier

3 VOSpace data model



3.1 Nodes and node types

We refer to the arrangement of data accessible through one particular VOspace service as “a VOspace”.

Each data object within a VOspace is represented as a node that is identified by a URI.

There are different types of nodes and the type of a VOspace node determines how the VOspace service stores and interprets the node data.

The types are arranged in a hierarchy, with more detailed types inheriting the structure of more generic types.

The following types are defined:

- *Node* is the most basic type
- *DataNode* describes a data item stored in the VOspace
- *UnstructuredDataNode* describes a data item for which the VOspace does not understand the data format

When data is stored and retrieved from an *UnstructuredDataNode*, the bit pattern read back shall be identical to that written.

- *StructuredDataNode* describes a data item for which the space understands the format and may make transformations that preserve the meaning of the data.

When data is stored and retrieved from a *StructuredDataNode*, the bit pattern returned may be different to the original. For example, storing tabular data from a VOTable file will preserve the tabular data, but any comments in the original XML file may be lost.

A *Node* has the following elements:

- *uri* : the `vos://` identifier for the node, URI-encoded according to RFC2396 [2]
- *properties* : a set of metadata properties for the node

A *DataNode* has the following elements:

- *accepts* : a list of the views (data formats) that the node can accept
- *provides* : a list of the views (data formats) that the node can provide
- *busy* : a boolean flag to indicate that the data associated with the node cannot be accessed

The *busy* flag is used to indicate that an internal operation is in progress, and the node data is not available.

In the current version of the specification, all nodes are either structured or unstructured data nodes.

Future versions of the specification may introduce new types of nodes.

The set of node types defined by this standard is closed; new types may be introduced only via new versions of the standard.

To comply with the standard, a client or service must be able to parse XML representations of all the node types defined in the current specification.

Note - This does not require all services to support all of the *Node* types, just that it can process an XML request containing any of the types. If the service receives a request for a type that it does not support, the service should return a *TypeNotSupported* fault. The service must not throw an XML parser error if it receives a request for a type that it does

not support.

3.2 Properties

Properties are simple string-based metadata properties associated with a node.

Individual *Properties* should contain simple short string values, not large blocks of information. If a system needs to attach a large amount of metadata to a node, then it should either use multiple small *Properties*, or a single *Property* containing a URI or URL pointing to an external resource that contains the additional metadata.

A *Property* has the following elements:

- *uri* : the *Property* identifier
- *value* : the string value of the *Property*
- *readOnly* : a boolean flag to indicate that the *Property* cannot be changed by the client

Properties may be set by the client or the service.

3.2.1 Property values

Unless they have special meaning to the service or client, *Properties* are treated as simple string values.

Some *Properties* may have meaning to the service. others may have meaning only to one specific type of client. A service implementation does not need to understand the meaning of all the *Properties* of a node. Any *Properties* that it does not understand can simply be stored as text strings.

3.2.2 Property identifiers

Every new type of *Property* requires a unique URI to identify the *Property* and its meaning.

The rules for the *Property* identifiers are similar to the rules for namespace URIs in XML schema. The only restriction is that it must be a valid (unique) URI.

- An XML schema namespace identifier can be just a simple URN, e.g. `urn:my-namespace`
- Within the IVOA, the convention for namespace identifiers is to use a HTTP URL pointing to the namespace schema or a resource describing it

The current VOSpace schema defines *Property* identifiers as *anyURI*. The only restriction is that it must be a valid (unique) URI.

- A *Property* URI can be a simple URN, e.g. `urn:my-property`

This may be sufficient for testing and development on a private system, but it is not scalable for use on a public service.

For a production system, any new *Properties* should have unique URIs that can be resolved into to a description of the *Property*.

Ideally, these should be IVO registry URIs that point to a description registered in the IVO registry :

- `ivo://my-registry/vospace/properties#my-property`

Using an IVO registry URI to identify *Properties* has two main advantages :

- IVO registry URIs are by their nature unique, which makes it easy to ensure that different teams do not accidentally use the same URI
- If the IVO registry URI points to a description registered in the IVO registry, this provides a mechanism to discover what the *Property* means

3.2.3 Property descriptions

If the URI for a particular *Property* is resolvable, i.e. an IVO registry identifier or a HTTP URL, then it should point to an XML resource that describes the *Property*.

A *Property* description should describe the data type and meaning of a *Property*.

A *PropertyDescription* should have the following members :

- *uri* : the formal URI of the *Property*
- *DisplayName* : A display name for the *Property*
- *Description* : A text block describing the meaning and validation rules of the *Property*

A *PropertyDescription* may have the following optional members :

- *UCD* : the Universal Content Descriptor (in the UCD1+ scheme) for the *Property*
- *Unit* : the unit of measurement of the *Property*

The information in a *Property* description can be used to generate a UI for displaying and modifying the different types of *Properties*.

Note that at the time of writing, the schema for registering *PropertyDescriptions* in the IVO registry has not been finalized.

3.2.3.1 UI display name

If a client is unable to resolve a *Property* identifier into a description, then it may just display the identifier as a text string :

- `urn:modified-date`

If the client can resolve the *Property* identifier into a description, then the client may use the information in the description to display a human readable name and description of the *Property* :

- *Last modification date of the node data*

3.2.3.2 Property editors

If the client is unable to resolve a *Property* identifier into a description, or does not understand the type information defined in the description, then the client may treat the *Property* value as a simple text string.

If the client can resolve the *Property* identifier into a description, then the client may use the information in the description to display an appropriate editing tool for the *Property*.

In the current version of the specification the rules for editing *Properties* are as follows :

- A service may impose validation rules on the values of specific types of *Properties*
- If a client attempts to set a *Property* to an invalid value, then the service may reject the change
- Where possible, the validation rules for a type of *Property* should be defined in the *Property* description

Future versions of the VOspace specification may extend the *PropertyDescription* to include more specific machine readable validation rules for a *Property* type.

Note that at the time of writing, the schema for registering validation rules in *PropertyDescriptions* has not been finalized.

3.2.4 Standard properties

The VOspace team intend to register *Property* URIs and *PropertyDescriptions* for the core set of *Properties*.

However, this is not intended to be a closed list, different implementations are free to define and use their own *Properties*.

3.3 Views

A *View* describes the data formats and contents available for importing or exporting data to or from a VOspace node.

The metadata for each VOspace *Node* contains two lists of *Views*.

- *accepts* is a list of *Views* that the service can accept for importing data into the *Node*
- *provides* is a list of *Views* that the service can provide for exporting data from *Node*

A *View* has the following members :

- *uri* : the *View* identifier
- *original* : an optional boolean flag to indicate that the *View* preserves the original bit pattern of the data
- *param* : a set of name-value pairs that can be used to specify additional arguments for the *View*

3.3.1 Example use cases

Simple file store

A simple VOspace system that stores data as a binary files can just return the contents of the original file. The client supplies a *View* identifier when it imports the data, and the service uses this information to describe the data to other clients.

A file based system can use the special case identifier

'`ivo://ivoa.net/vospace/core#view-any`' to indicate that it will accept any data format or *View* for a *Node*.

For example :

- A client imports a file into the service, specifying a *View* to describe the file contents

- The service stores the data as a binary file and keeps a record of the *View*
- The service can then use the *View* supplied by the client to describe the data to other clients

This type of service is not required to understand the imported data, or to verify that its contents match the *View*, it treats all data as binary files.

3.3.1.1 Database store

A VOspace system that stores data in database tables would need to be able to understand the data format of an imported file in order to parse the data and store it correctly. This means that the service can only accept a specific set of *Views*, or data formats, for importing data into the *Node*.

In order to tell the client what input data formats it can accept, the service publishes a list of specific *Views* in the *accepts* list for each *Node*.

On the output side, a database system would not be able to provide access to the original input file. The contents of file would have been transferred into the database table and then discarded. The system has to generate the output results from the contents of the database table.

In order to support this, the service needs to be able to tell the client what *Views* of the data are available.

A database system may offer access to the table contents as either VOTable or FITS files, it may also offer zip or tar.gz compressed versions of these. In which case the system needs to be able to express nested file formats such as 'zip containing VOTable' and 'tar.gz containing FITS'.

A service may also offer subsets of the data. For example, a work flow system may only want to look at the table headers to decide what steps are required to process the data. If the table contains a large quantity of data, then downloading the whole contents just to look at the header information is inefficient. To make this easier, a database system may offer a 'metadata only' *View* of the table, returning a VOTable or FITS file containing just the metadata headers and no rows.

So our example service may want to offer the following *Views* of a database table :

- Table contents as FITS
- Table contents as VOTable
- Table contents as zip containing FITS
- Table contents as zip containing VOTable
- Table contents as tar.gz containing FITS
- Table contents as tar.gz containing VOTable
- Table metadata as FITS
- Table metadata as VOTable

The service would publish this information as a list of *Views* in the *provides* section of the metadata for each *Node*.

The VOspace specification does not mandate what *Views* a service must provide. The VOspace specification is intended to provide a flexible mechanism enabling services to describe a variety of different *Views* of data. It is up to the service implementation to

decide what *Views* of the data it can accept and provide.

3.3.2 View identifiers

Every new type of *View* requires a unique URI to identify the *View* and its content.

The rules for the *View* identifiers are similar to the rules for namespace URIs in XML schema. The only restriction is that it must be a valid (unique) URI.

- An XML schema namespace identifier can be just a simple URN, e.g. `urn:my-namespace`
- Within the IVOA, the convention for namespace identifiers is to use a HTTP URL pointing to the namespace schema, or a resource describing it

The current VOSpace schema defines *View* identifiers as *anyURI*. The only restriction is that it must be a valid (unique) URI.

- A *View* URI can be a simple URN, e.g. `urn:my-view`

This may be sufficient for testing and development on a private system, but it is not scalable for use on a public service.

For a production system, any new *Views* should have unique URIs that can be resolved into to a description of the *View*.

Ideally, these should be IVO registry URIs that point to a description registered in the IVO registry :

- `ivo://my-registry/vospace/views#my-view`

Using an IVO registry URI to identify *Views* has two main advantages :

- IVO registry URIs are by their nature unique, which makes it easy to ensure that different teams do not accidentally use the same URI
- If the IVO registry URI points to a description registered in the IVO registry, this provides a mechanism to discover what the *View* contains

3.3.3 View descriptions

If the URI for a particular *View* is resolvable, i.e. an IVO registry identifier or a HTTP URL, then it should point to an XML resource that describes the *View*.

A *ViewDescription* should describe the data format and/or content of the view.

A *ViewDescription* should have the following members :

- *uri* : the formal URI of the *View*
- *DisplayName* : A simple text display name of the *View*
- *Description* : Text block describing the data format and content of the *View*

A *ViewDescription* may have the following optional members :

- *MimeType* : the standard MIME type of the *View*, if applicable

However, at the time of writing, the schema for registering *ViewDescriptions* in the IVO registry has not been finalized.

3.3.3.1 UI display name

If a client is unable to resolve a *View* identifier into a description, then it may just display the identifier as a text string :

- *Download as* urn:table.meta.fits

If the client can resolve the *View* identifier into a description, then the client may use the information in the description to display a human readable name and description of the *View* :

- *Download table metadata as FITS header*

3.3.3.2 MIME types

If a VOSpace service provides HTTP access to the data contained in a *Node*, then if the *ViewDescription* contains a *MimeType* field, this should be included in the appropriate header field of the HTTP response.

3.4 Protocols

A *Protocol* describes the parameters required to perform a data transfer using a particular protocol.

A *Protocol* has the following members :

- *uri* : the *Protocol* identifier
- *endpoint* : the endpoint URL to use for the data transfer
- *param* : A list of name-value pairs that specify any additional arguments required for the transfer

3.4.1 Protocol identifiers

Every new type of *Protocol* requires a unique URI to identify the *Protocol* and how to use it.

The rules for the *Protocol* identifiers are similar to the rules for namespace URIs in XML schema. The only restriction is that it must be a valid (unique) URI

- An XML schema namespace identifier can be just a simple URN, e.g. urn:my-namespace
- Within the IVOA, the convention for namespace identifiers is to use a HTTP URL pointing to the namespace schema, or a resource describing it

The current VOSpace schema defines *Protocol* identifiers as *anyURI*. The only restriction is that it must be a valid (unique) URI.

- A *Protocol* URI can be a simple URN, e.g. urn:my-protocol

This may be sufficient for testing and development on a private system, but it is not scalable for use on a public service.

For a production system, any new *Protocols* should have unique URIs that can be resolved into to a description of the *Protocol*.

Ideally, these should be IVO registry URIs that point to a description registered in the IVO registry :

- `ivo://my-registry/vospace/protocols#my-protocol`

Using an IVO registry URI to identify *Protocols* has two main advantages :

- IVO registry URIs are by their nature unique, which makes it easy to ensure that different teams do not accidentally use the same URI
- If the IVO registry URI points to a description registered in the IVO registry, this provides a mechanism to discover how to use the *Protocol*

3.4.2 Protocol descriptions

If the URI for a particular *Protocol* is resolvable, i.e. an IVO registry identifier or a HTTP URL, then it should point to an XML resource that describes the *Protocol*.

A *ProtocolDescription* should describe the underlying transport protocol, and how it should be used in this context.

A *ProtocolDescription* should have the following members :

- *uri* : the formal URI of the *Protocol*
- *DisplayName* : A simple display name of the *Protocol*
- *Description* : Text block describing describing the underlying transport protocol, and how it should be used in this context

However, at the time of writing, the schema for registering *ProtocolDescriptions* in the IVO registry has not been finalized.

3.4.2.1 UI display name

If a client is unable to resolve a *Protocol* identifier into a description, then it may just display the identifier as a text string :

- *Download using* `urn:my-protocol`

If the client can resolve the *Protocol* identifier into a description, then the client may use the information in the description to display a human readable name and description of the *Protocol* :

- *Download using standard HTTP GET*

3.4.3 Standard protocols

The VOSpace team intend to register *Protocol* URIs and *ProtocolDescriptions* for the core set of standard transport protocols.

e.g.

- Standard HTTP get and put
- Standard FTP get and put

However, this is not intended to be a closed list, different implementations are free to define and use their own transfer *Protocols*.

3.4.4 Custom protocols

There are several use cases where a specific VOSpace implementation may want to

define and use a custom VOSpace transfer *Protocol*, either extending an existing *Protocol*, or defining a new one.

3.4.4.1 SRB gateway

One example would be a VOSpace service that was integrated with a SRB system. In order to enable the service to use the native SRB transport protocol to transfer data, the service providers would need to register a new *ProtocolDescription* to represent the SRB transport protocol.

The *ProtocolDescription* would refer to the technical specification for the SRB transport protocol, and define how it should be used to transfer data to and from the VOSpace service.

Clients that do not understand the SRB transport protocol would not recognize the URI for the SRB *Protocol*, and would ignore *Transfer* options offered using this *Protocol*.

Clients that were able to understand the SRB transport protocol would recognize the URI for the SRB *Protocol*, and could use the 'srb://..' endpoint address in a *Protocol* option to transfer data using the SRB transport protocol.

Enabling different groups to define, register and use their own custom *Protocols* in this way means that support for new transport protocols can be added to VOSpace systems without requiring changes to the core VOSpace specification.

In this particular example, it is expected that one group within the IVOA will work with the SRB team at SDSC to define and register the *Protocol* URI and *ProtocolDescription* for using the SRB protocol to transfer data to and from VOSpace systems.

Other implementations that plan to use the SRB transport protocol in the same way could use the same *Protocol* URI and *ProtocolDescription* to describe data transfers using the SRB transport protocol.

The two implementations would then be able use the common *Protocol* URI to negotiate data transfers using the SRB transport protocol.

3.4.4.2 Local NFS transfers

Another example of a custom *Protocol* use case would to transfer data using the local NFS file system within an institute.

If an institute has one or more VOSpace services co-located with a number of data processing applications, all located within the same local network, then it would be inefficient to use HTTP get and put to transfer the data between the services if they could all access the same local file system.

In this case, the local system administrators could register a custom *ProtocolDescription* which described how to transfer data using their local NFS file system.

- `ivo://my.institute/vospace/protocols#internal-nfs`

Data transfers using this *Protocol* would be done using `file://` URLs pointing to locations within the local NFS file system :

- `file:///net/host/path/file`

These URLs would only have meaning to services and applications located within the local

network, and would not be usable from outside the network.

Services and applications located within the local network would be configured to recognize the custom *Protocol* URI, and to use local file system operations to move files within the NFS file system.

Services and applications located outside local network would not recognize the custom *Protocol* URI and so would not attempt to use the internal file URLs to transfer data.

Note that in this example the custom *Protocol* URI and the associated *ProtocolDescription* refer to data transfers using file URLs **within a specific local NFS file system**.

If a different institute wanted to use a similar system to transfer data within their own local network, then they would have to register their own custom *Protocol* URI and associated *ProtocolDescription*.

The two different *Protocol* URIs and *ProtocolDescriptions* describe how to use the same underlying transport protocol (NFS) in different contexts.

Enabling different groups to define, register and use their own custom *Protocols* in this way means that systems can be configured to use the best available transport protocols for transferring data, without conflicting with other systems who may be using similar a transport protocol in a different context.

3.5 Transfers

A *Transfer* describes the details of a data transfer to or from a space.

A *Transfer* has the following members :

- *view* : A *View* specifying the requested *View*
 - For the transfer to be valid, the specified *View* must match one of those listed in the *accepts* or *provides* list of the *Node*
- *protocol* : one or more a *Protocols* specifying the transfer protocols to use
 - A *Transfer* may contain more than one *Protocol* element with different *Protocol* URIs
 - A *Transfer* may contain more than one *Protocol* element with the same *Protocol* URI with different endpoints

3.5.1 Synchronous transfers

Two of the VOSpace transfer methods are synchronous - the service performs the data transfer during the scope of the SOAP call.

In these methods, the client constructs a *Transfer* request containing details of the *Node* and *View* and one or more *Protocol* elements with valid endpoint addresses.

The service may ignore *Protocols* with URIs that it does not recognize.

If the server is unable to handle any of the requested *Protocols* in a *Transfer* request, then it responds with a fault.

The order of the *Protocols* in the request indicates the order of preference that the client would like the server to use. However, this is only a suggestion, and the server is free to use its own preferences to select which *Protocol* it uses first.

The service selects the first *Protocol* it wants to use from the list and attempts to transfer the data using the *Protocol* endpoint.

If the first attempt fails, the server may choose another *Protocol* from the list and re-try the transfer using that *Protocol* endpoint.

The server may attempt to transfer the data using any or all of the *Protocols* in the list until either, the data transfer succeeds, or there are no more *Protocol* options left.

The server is only allowed to use each *Protocol* option once. This allows a data source to issue one time URLs for a *Transfer*, and cancel each URL once it has been used.

Once one of the *Protocol* options succeeds the transfer is considered to be completed, and the server is not allowed to use any of the remaining *Protocol* options. This allows a data source to issue a set of one time URLs for a transfer, and to cancel any unused URLs once the transfer has been completed.

Some *Protocols* may require the service to call a callback address when a data transfer completes. This behavior is specific to the *Protocol*, and should be defined in the *ProtocolDescription*.

If none of the *Protocol* options succeed, then the transfer is considered to have failed, and the service returns an exception containing details of the *Protocol* options it tried.

3.5.2 Asynchronous transfers

Two of the VOSpace transfer methods are asynchronous - an external actor performs the data transfer outside the scope of the SOAP call.

In these methods, the client sends a template *Transfer* request to the server.

The *Transfer* request should contain details of the *Node* and *View* and one or more *Protocol* elements without endpoint addresses.

In effect, the client is sending a list of *Protocols* that it (the client) wants to use for the transfer.

The service may ignore *Protocols* with URIs that it does not recognize.

The service selects the *Protocols* from the request that it is capable of handling, and builds a *Transfer* response containing the selected *Protocol* elements filling in valid endpoint addresses for each of them.

The order of the *Protocol* elements in the request indicates the order of preference that the client would like to use. However, this is only a suggestion, and the server is free to use its own preferences when generating the list of *Protocols* in the response.

In effect, the server is responding with the subset of the requested *Protocols* that it (the server) is prepared to offer.

If the server is unable to accept any of the requested *Protocols*, then it responds with a fault.

On receipt of the response, the client can use the list of *Protocols* itself, or pass them on to another agent to perform the data transfer on its behalf.

The agent may ignore *Protocols* URIs that it does not recognize.

The agent selects the first *Protocol* it wants to use from the list and attempts to transfer the

data using the *Protocol* endpoint.

If the first attempt fails, the agent may choose another *Protocol* from the list and re-try the transfer using that *Protocol* endpoint.

The agent may attempt to transfer the data using any or all of the *Protocols* in the list until either, the data transfer succeeds, or there are no more *Protocol* options left.

The agent is only allowed to use each *Protocol* option once. This allows a data source to issue one time URLs for a *Transfer*, and cancel each URL once it has been used.

Once one of the *Protocol* options succeeds the transfer is considered to be completed, and the agent is not allowed to use any of the remaining unused *Protocol* options. This allows a data source to issue a set of one time URLs for a transfer, and to cancel any unused URLs once the transfer has been completed.

Some *Protocols* may require the agent to call a callback address when a data transfer completes. This behavior is specific to the *Protocol*, and should be defined in the *ProtocolDescription*.

If none of the *Protocol* options succeed, then the transfer is considered to have failed.

4 Access control

The access control policy for a VOSpace is defined by the implementor of that space according to the use cases for which the implementation is intended.

A human-readable description of the implemented access policy must be declared in the registry metadata for the space.

These are the most probable access policies :

- No access control is asserted. Any client can create, read, write and delete nodes anonymously
- No authorization is required, but clients must authenticate an identity (for logging purposes) in each request to the space
- Clients may not create or change nodes (i.e. the contents of the space are fixed by the implementation or set by some interface other than VOSpace), but any client can read any node without authentication
- Nodes are considered to be owned by the user who created them. Only the owner can operate on a node

No operations to modify the access policy (e.g. to set permissions on an individual node) are included in this standard. Later versions may add such operations.

Where the access policy requires authentication of callers, the VOSpace service shall support one of the authentication methods defined in the IVOA Single Sign On profile.

5 Web service operations

A VOSpace-1.0 service shall be a SOAP service with the following operations.

5.1 Service metadata

5.1.1 getProtocols

Get a list of the transfer *Protocols* supported by the space service.

5.1.1.1 Parameters

- none

5.1.1.2 Returns

- *accepts* : A list of *Protocols* that the service can accept
 - In this context 'accepting a protocol' means that the service can act as a client for the protocol
 - e.g. 'accepting http-get' means the service can read data from an external http web server
- *provides* : A list of *Protocols* that the service can provide
 - In this context 'providing a protocol' means that the service can act as a server for the protocol
 - e.g. 'providing http-get' means the service can act as a http web server

5.1.1.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails

5.1.2 getViews

Get a list of the *Views* and data formats supported by the space service.

5.1.2.1 Parameters

- none

5.1.2.2 Returns

- *accepts* : A list of *Views* that the service can accept
 - In this context 'accepting a view' means that the service can receive input data in this format
 - A simple file based system may use the reserved *View* URI `ivo://net.ivoa.vospace/views/any` to indicate that it can accept data in any format
- *provides* : A list of *Views* that the service can provide
 - In this context 'providing a view' means that the service can produce output data in this format
 - A simple file based system may use the reserved *View* URI `ivo://net.ivoa.vospace/views/any` to indicate that it can provide data in any format

5.1.2.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails

5.1.3 getProperties

5.1.3.1 Parameters

- none

5.1.3.2 Returns

- *accepts* : A list of identifiers for the *Properties* that the service accepts and understands. This refers to metadata (*Properties*) that is implementation dependent but can be used by a client to control operational aspects of the service: for example, a VOSpace implementation might allow individual users to control the permissions on data objects via a *Property* called "permissions". If the VOSpace receives a data object with this *Property* then it 'understands' what this property refers to and can deal with it accordingly.
- *provides* : A list of identifiers for the *Properties* that the service provides
- *contains* : A list of identifiers for all the *Properties* currently used by *Nodes* within the service

5.1.3.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails

5.2 Creating and manipulating data nodes

5.2.1 createNode

Create a new node at a specified location.

5.2.1.1 Parameters

- *node* : A template *Node* (as defined in Section 3.1) for the node to be created

A valid *uri* attribute is required. If the reserved URI `vos://null` is used the service will replace it with a new unique service-generated URI.

If the *Node* `xsi:type` is not specified then a generic node of type *Node* is implied.

The permitted values of `xsi:type` are :

- `vos:Node`
- `vos:DataNode`
- `vos:UnstructuredDataNode`
- `vos:StructuredDataNode`

When creating a new *Node* the service may substitute a valid subtype, i.e. If `xsi:type` is set to `vos:DataNode` then this may be implemented as a *DataNode*, *StructuredDataNode* or an *UnstructuredDataNode*.

The properties of the new *Node* can be set by adding *Properties* to the template. Attempting to set a *Property* that the service considers to be 'readonly' will cause a *PermissionDenied* exception.

The *accepts* and *provides* lists of *Views* for the *Node* cannot be set using this method.

5.2.1.2 Returns

- *node* : details of the new *Node*

The *accepts* list of *Views* for the *Node* will be filled in by the service based on service capabilities.

The *provides* list of *Views* for the *Node* may not be filled in until some data has been imported into the *Node*.

5.2.1.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *DuplicateNode* exception if a *Node* already exists with the same URI
- The service shall throw an *InvalidURI* exception if the requested URI is invalid
- The service shall throw a *TypeNotSupported* exception if the type specified in `xsi:type` is not supported
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation

5.2.2 deleteNode

Delete a node.

5.2.2.1 Parameters

- *target* : the URI of an existing *Node*

5.2.2.2 Returns

- *void*

5.2.2.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if the target node does not exist

5.2.3 listNodes

List nodes in the space.

In order to support large numbers of nodes, this method uses a continuation token to enable the list of results to be split across more than response.

5.2.3.1 Parameters

- *token* : An optional continuation token from a previous request
 - No token indicates a request for a new list

The server may impose a limited lifetime on the continuation token. If a token has expired, the server will throw an exception, and the client will have to make a new request.

- *limit* : An optional limit indicating the maximum number of results in the response
 - No limit indicates a request for an unpaginated list. However the server may still impose its own limit on the size of an individual response, splitting the results into more than one page if required
- *detail* : The level of detail in the returned listing
 - *min* : The list contains the minimum detail for each *Node* with all optional parts removed – the node type should be returned
 - e.g. `<node uri="vos://service/name" xsi:type="Node"/>`
 - *max* : The list contains the maximum detail for each *Node*, including any `xsi:type` specific extensions
 - *properties* : The list contains a basic *node* element with a list of *properties* for each *Node* with no `xsi:type` specific extensions.
- *nodes* : A list containing zero or more *Nodes* identifying the target URIs to be listed

Only the *uri* of the *Nodes* in the request list are used, the service will ignore any other elements or attributes. This method does not perform a search based on the *Node Properties* or other attributes.

The *Node* URIs in the request list may contain a '*' wild card character in the *name* part of the URI (the remaining text following the last '/' character).

A single request may include more than one target *Node* containing a wild card. For example, the following request lists all *Nodes* with names that match either '*.xml' or '*.txt'

```
<listNodes>
  <nodes>
    <node uri="vos://service/*.xml"/>
    <node uri="vos://service/*.txt"/>
  </nodes>
</listNodes>
```

Note that the wild card substitution for the '*' is a simple 'zero or more of any characters' match.

So a request for '*.txt' will match *Nodes* with the the following names :

- .txt
- frog.txt

This method does not support regular expression matches.

An empty list of target *Nodes* list implies a full listing of the space.

A request with an empty list of target *Nodes* :

```
<listNodes>
  <nodes/>
</listNodes>
```

is equivalent to a request with a single target *Node* URI of '*' :

```
<listNodes>
  <nodes>
    <node uri="vos://service/*"/>
  </nodes>
</listNodes>
```

5.2.3.2 Returns

- *token* : An optional continuation token, indicating that the list is incomplete
 - The client may use this token to request the next block of *Nodes* in the sequence
 - No token indicates that the list is complete.
- *limit* : An optional limit which must be present if a limit parameter was used in the request
 - If present, the value is the value from the original request and not any limit imposed by the service
- *nodes* : A list of the *Nodes* matching the requested *Node* URIs

5.2.3.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if a specific target *Node* does not exist
 - This does not apply if the target *Node* URI contains a wild card
- The service shall throw an *InvalidToken* exception if it does not recognize the continuation token
- The service shall throw an *InvalidToken* exception if the continuation token has expired

5.2.4 moveNode

Move a node within a VOSpace service.

Note that this does not cover moving data between two separate VOSpace services.

Moving nodes between separate VOSpace services should be handled by the client, using the import, export and delete methods.

5.2.4.1 Parameters

- *source* : The URI of an existing *Node*
- *destination* : A template describing the new *Node* (as defined in section 3.1).

If the *destination uri* is set to the reserved URI `vos://null` then the service will generate a new unique URI for the *Node*.

The *Properties* from the *source Node* will be combined with the *Properties* from the *destination* to create the new *Node*.

The *Node* type cannot be changed using this method.

The value of the `xsi:type` attribute on the *destination* will be ignored and the new *Node* will inherit the same type as the original.

The *accepts* and *provides* lists of *Views* for the new *Node* cannot be set using this method.

5.2.4.2 Returns

- *node* : Details of the *Node* in its new location

5.2.4.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if the source *Node* does not exist
- The service shall throw a *DuplicateNode* exception if a *Node* already exists at the destination
- The service shall throw an *InvalidURI* exception if a specified URI is invalid
- The service shall throw an *InvalidArgument* exception if a specified value is invalid

5.2.5 copyNode

Copy a node within a VOSpace service.

Note that this does not cover copying data between two separate VOSpace services.

Copying nodes between separate VOSpace services should be handled by the client, using the import and export methods.

5.2.5.1 Parameters

- *source* : The URI of an existing *Node*
- *destination* : A template *Node* (as defined in section 3.1) describing the new *Node*

If the *destination uri* is set to the reserved URI `vos://null` then the service will generate a new unique URI for the *Node*.

The *Properties* from the *source Node* will be combined with the *Properties* from request to create the new *Node*.

The *Node* type cannot be changed using this method.

The value of the `xsi:type` attribute on the *destination* will be ignored and the new *Node* will inherit the same type as the original.

The *accepts* and *provides* lists of *Views* for the new *Node* cannot be set using this method.

5.2.5.2 Returns

- *node* : Details of the new *Node*

5.2.5.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if the source *Node* does not exist
- The service shall throw a *DuplicateNode* exception if a *Node* already exists at the destination
- The service shall throw an *InvalidURI* exception if a specified URI is invalid
- The service shall throw an *InvalidArgument* exception if a specified value is invalid

5.3 Accessing metadata

5.3.1 getNode

Get the details for a specific *Node*.

5.3.1.1 Parameters

- *target* : The URI of an existing *Node*

5.3.1.2 Returns

- *node* : Details of the *Node*

The full expanded record for the node is returned, including any `xsi:type` specific extensions.

5.3.1.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if the target *Node* does not exist

5.3.2 setNode

Set the property values for a specific node.

5.3.2.1 Parameters

- *node* : A *Node* containing a the *Node uri* and a list of the *Properties* to set (as defined in section 3.1)

This will add or update the node properties including any `xsi:type` specific elements.

The operation is the union of existing values and new ones.

- An empty value sets the value to blank.
- To delete a *Property*, set the `xsi:nil` attribute to `true`

This method cannot be used to modify the *Node* type.

This method cannot be used to modify the *accepts* or *provides* list of *Views* for the *Node*.

5.3.2.2 Returns

- *node* : Details of the *Node*

The full expanded record for the node is returned, including any `xsi:type` specific extensions.

5.3.2.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the request attempts to modify a *readonly Property*
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw a *NodeNotFound* exception if the target *Node* does not exist
- The service shall throw an *InvalidArgument* exception if a specified *property* value is invalid

5.4 Transferring data

5.4.1 pushToVoSpace

Request a list of URLs to send data to a VOSpace node.

This method asks the server for a list of one or more URLs that the client can use to send data to.

The data transfer is initiated by the client, after it has received the response from the VOSpace service.

The primary use case for this method is client that wants to send some data directly to a VOSpace service.

5.4.1.1 Parameters

- *destination* : A description of the target *Node* (as defined in section 3.1)

A valid *uri* attribute is required.

If a *Node* already exists at the target URI, then the data will be imported into the existing *Node* and the *Node properties* will be updated with the *Properties* in the request.

If there is no *Node* at the destination URI, then the service will create a new *Node* using the *uri*, *xsi:type* and *properties* supplied in the request (see *createNode* for details).

- *transfer* : A template (as defined in section 3.5) for the data *Transfer*

The *Transfer* template contains details of the *View* and a list of the *Protocols* that the client wants to use.

The list of *Protocols* should not contain *endpoint* addresses, the service will supply the *endpoint* addresses in the response.

The service will ignore any of the requested protocols that it does not understand or is unable to support.

5.4.1.2 Returns

- *destination* : Updated details of the destination *Node*
- *transfer* : Updated details of the data *Transfer*

The service selects which of the requested *Protocols* it is willing to provide and fills in the operational details for each one – this will normally include specifying the destination URL of the transfer protocol endpoint.

The service response should not include any *Protocols* which it is unable to support.

5.4.1.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service will throw a *TypeNotSupported* exception if it is unable to create a new *Node* of the requested type
- The service shall throw a *ViewNotSupported* exception if a *StructuredDataNode* is requested with no *View*
- The service shall throw a *ViewNotSupported* exception if the service does not support the requested *View*
- The service shall throw a *ProtocolNotSupported* exception if it supports none of the requested *Protocols*
- The service shall throw an *InvalidURI* exception if the *destination* URI is invalid
- The service shall throw an *InvalidArgument* exception if a *View* parameter is invalid
- The service shall throw an *InvalidArgument* exception if a *Protocol* parameter is invalid

5.4.2 pullToVoSpace

Import data into a VOSpace node.

This method asks the server to fetch data from a remote location.

The data transfer is initiated by the VOSpace service and transferred direct into the target *Node*.

The data source can be another VOSpace service, or a standard HTTP or FTP server. The primary use case for this method is transferring data from one server or service to another.

5.4.2.1 Parameters

- *destination* : A description of the target *Node* (as defined in section 3.1)

A valid *uri* attribute is required.

If a *Node* already exists at the target URI, then the data will be imported into the existing *Node* and the *Node properties* will be updated with the *Properties* in the request.

If there is no *Node* at the destination URI, then the service will create a new *Node* using the *uri*, *xsi:type* and *properties* supplied in the request (see *createNode* for details).

- *transfer* : Details of the *Transfer* (as defined in section 3.5)

The *Transfer* details should include the *View* and a list of one or more *Protocols* with valid *endpoints* and parameters. The endpoint will normally refer to the source URL for the transfer protocol.

5.4.2.2 Returns

- *destination* : Updated details of the destination *Node*

5.4.2.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service will throw a *TypeNotSupported* exception if it is unable to create a new *Node* of the requested type
- The service shall throw a *ViewNotSupported* exception if a *StructuredDataNode* is requested with no *View*
- The service shall throw a *ViewNotSupported* exception if the service does not support the requested *View*
- The service shall throw a *ProtocolNotSupported* exception if it supports none of the requested *Protocols*
- The service shall throw an *InvalidURI* exception if the destination URI is invalid
- The service shall throw an *InvalidArgument* exception if a *View* parameter is invalid
- The service shall throw an *InvalidArgument* exception if a *Protocol* parameter is invalid
- The service shall throw a *TransferFailed* exception if the data transfer does not complete
- The service shall throw an *InvalidData* exception if the data does not match the *View*

5.4.2.4 Notes

In VOSpace version 1.0, the transfer is synchronous, and the SOAP call does not return until the transfer has been completed.

If the *Transfer* request contains more than one *Protocol* option, then the service may fail over to use one or more of the options if the first one fails. The service should try each *Protocol* option in turn until one succeeds or all have been tried.

5.4.3 pullFromVoSpace

Request set of URLs that the client can read data from.

The client requests access to the data in a *Node*, and the server responds with a set of URLs that the client can read the data from.

5.4.3.1 Parameters

- *source* : The URI of an existing *DataNode*
- *transfer* : A template for the *Transfer* (as defined in section 3.5)

The template for the *Transfer* should contain details of the *View* and a list of the *Protocols* that the client would like to use.

The list of *Protocols* should not contain *endpoint* addresses, the service will supply the *endpoint* addresses in the response.

The service will ignore any of the requested protocols that it does not understand or is unable to support.

5.4.3.2 Returns

- *transfer* : Updated details of the data *Transfer*

The service selects which of the requested *Protocols* it is willing to provide and fills in the operational details for each one – this will normally include specifying the source URL of the transfer protocol endpoint.

The service response should not include any *Protocols* which it is unable to support.

5.4.3.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw an *InvalidURI* exception if the *source* URI is invalid
- The service shall throw a *NodeNotFound* exception if the source *Node* does not exist.
- The service shall throw a *ProtocolNotSupported* exception if it none of the requested *Protocols* are supported
- The service shall throw a *ViewNotSupported* exception if it does not support the requested *View*
- The service shall throw an *InvalidArgument* exception if a *View* parameter is invalid
- The service shall throw an *InvalidArgument* exception if *Protocols* a parameter is invalid

5.4.3.4 Notes

The endpoint URLs supplied in the response should be considered as 'one shot' URLs. A

VOspace service connected to a standard web server may return the public URL for the data. However, a different implementation may create a unique single use URL specifically for this transfer.

5.4.4 pushFromVoSpace

Ask the server to send data to a remote location.

The client supplies a list of URLs and asks the server to send the data to the remote location.

The transfer is initiated by the server, and the data is transferred direct from the server to the remote location.

5.4.4.1 Parameters

- *source* : The URI of an existing *DataNode*
- *transfer* : Details of the *Transfer* (as defined in section 3.5)

The *Transfer* details should include the *View* and a list of one or more *Protocols* with valid *endpoint* and *parameters*. The endpoints will normally refer to the destination URLs for the transfer protocols.

5.4.4.2 Returns

- void

5.4.4.3 Faults

- The service shall throw an *InternalFault* exception if the operation fails
- The service shall throw a *PermissionDenied* exception if the user does not have permissions to perform the operation
- The service shall throw an *InvalidURI* exception if the *source* URI is invalid
- The service shall throw a *NodeNotFound* exception if the source *Node* does not exist
- The service shall throw a *ProtocolNotSupported* exception if it supports none of the requested *Protocols*
- The service shall throw a *ViewNotSupported* exception if it does not support the requested *View*
- The service shall throw an *InvalidArgument* exception if a *Protocols* parameter is invalid
- The service shall throw a *TransferFailed* exception if the data transfer does not complete

5.4.4.4 Notes

In VOspace version 1.0, the transfer is synchronous, and the SOAP call does not return until the transfer has been completed.

If the *Transfer* request contains more than one *Protocol* then the service may fail over to use one or more of the options if the first one fails. The service should try each *Protocol* option in turn until one succeeds or all have been tried.

5.5 Fault arguments

5.5.1 InternalFault

This is thrown with a description of the cause of the fault.

5.5.2 PermissionDenied

This is thrown with no arguments.

5.5.3 InvalidURI

This is thrown with details of the invalid URI.

5.5.4 NodeNotFound

This is thrown with the URI of the missing *Node*.

5.5.5 DuplicateNode

This is thrown with the URI of the duplicate *Node*.

5.5.6 InvalidToken

This is thrown with the invalid token.

5.5.7 InvalidArgument

This is thrown with a description of the invalid argument, including the View or Protocol URI and the name and value of the parameter that caused the fault.

5.5.8 TypeNotSupported

This is thrown with the *QName* of the unsupported type.

5.5.9 ViewNotSupported

This is thrown with the uri of the *View*.

5.5.10 InvalidData

This is thrown with any error message that the data parser produced.

6 Appendix: Machine readable definitions

6.1 WSDL

```

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:vos.contract.1.0="http://www.net.ivoa/xml/VOSpaceContract-v1.0"
  xmlns:vos.types.1.0="http://www.net.ivoa/xml/VOSpaceTypes-v1.0"
  xmlns="http://www.net.ivoa/xml/VOSpaceContract-v1.0"
  targetNamespace="http://www.net.ivoa/xml/VOSpaceContract-v1.0"
  >
  <wsdl:documentation>
    Interface definition for a VOSpace 1.0 web service
  </wsdl:documentation>
  <wsdl:types>

    <xsd:schema
      targetNamespace="http://www.net.ivoa/xml/VOSpaceContract-v1.0"
      elementFormDefault="qualified"
      >

      <!--+
        | Import the message schema.
      +-->
      <xsd:import
        namespace="http://www.net.ivoa/xml/VOSpaceTypes-v1.0"
        schemaLocation="VOSpaceTypes-v1.0.xsd"
      />

      <!--+
        | The GetViews message elements.
      +-->
      <xsd:element name="GetViews"          type="vos.types.
1.0:GetViewsRequestType"/>

```

```

        <xsd:element name="GetViewsResponse" type="vos.types.
1.0:GetViewsResponseType"/>

        <!--+
          | The GetProtocols message elements.
        +-->

        <xsd:element name="GetProtocols"           type="vos.types.
1.0:GetProtocolsRequestType"/>

        <xsd:element name="GetProtocolsResponse" type="vos.types.
1.0:GetProtocolsResponseType"/>

        <!--+
          | The GetProperties message elements.
        +-->

        <xsd:element name="GetProperties"           type="vos.types.
1.0:GetPropertiesRequestType"/>

        <xsd:element name="GetPropertiesResponse" type="vos.types.
1.0:GetPropertiesResponseType"/>

        <!--+
          | The CreateNode message elements.
        +-->

        <xsd:element name="CreateNode"           type="vos.types.
1.0:CreateNodeRequestType"/>

        <xsd:element name="CreateNodeResponse" type="vos.types.
1.0:CreateNodeResponseType"/>

        <!--+
          | The DeleteNode message elements.
        +-->

        <xsd:element name="DeleteNode"           type="vos.types.
1.0>DeleteNodeRequestType"/>

        <xsd:element name="DeleteNodeResponse" type="vos.types.
1.0>DeleteNodeResponseType"/>

        <!--+
          | The MoveNode message elements.
        +-->

        <xsd:element name="MoveNode"           type="vos.types.
1.0:MoveNodeRequestType"/>

        <xsd:element name="MoveNodeResponse" type="vos.types.
1.0:MoveNodeResponseType"/>

        <!--+
          | The CopyNode message elements.

```

```

        +-->
        <xsd:element name="CopyNode"          type="vos.types.
1.0:CopyNodeRequestType"/>
        <xsd:element name="CopyNodeResponse" type="vos.types.
1.0:CopyNodeResponseType"/>

<!--+
    | The GetNode message elements.
    +-->
    <xsd:element name="GetNode"              type="vos.types.
1.0:GetNodeRequestType"/>
    <xsd:element name="GetNodeResponse"     type="vos.types.
1.0:GetNodeResponseType"/>

<!--+
    | The SetNode message elements.
    +-->
    <xsd:element name="SetNode"              type="vos.types.
1.0:SetNodeRequestType"/>
    <xsd:element name="SetNodeResponse"     type="vos.types.
1.0:SetNodeResponseType"/>

<!--+
    | The ListNodes message elements.
    +-->
    <xsd:element name="ListNodes"           type="vos.types.
1.0:ListNodesRequestType"/>
    <xsd:element name="ListNodesResponse"   type="vos.types.
1.0:ListNodesResponseType"/>

<!--+
    | The PushToVoSpace message elements.
    +-->
    <xsd:element name="PushToVoSpace"       type="vos.types.
1.0:PushToVoSpaceRequestType"/>
    <xsd:element name="PushToVoSpaceResponse" type="vos.types.
1.0:PushToVoSpaceResponseType"/>

<!--+
    | The PullToVoSpace message elements.
    +-->
    <xsd:element name="PullToVoSpace"       type="vos.types.
1.0:PullToVoSpaceRequestType"/>
    <xsd:element name="PullToVoSpaceResponse" type="vos.types.
1.0:PullToVoSpaceResponseType"/>

```

```

<!--+
  | The PullFromVoSpace message elements.
  +-->
  <xsd:element name="PullFromVoSpace"          type="vos.types.
1.0:PullFromVoSpaceRequestType"/>
  <xsd:element name="PullFromVoSpaceResponse" type="vos.types.
1.0:PullFromVoSpaceResponseType"/>

<!--+
  | The PushFromVoSpace message elements.
  +-->
  <xsd:element name="PushFromVoSpace"          type="vos.types.
1.0:PushFromVoSpaceRequestType"/>
  <xsd:element name="PushFromVoSpaceResponse" type="vos.types.
1.0:PushFromVoSpaceResponseType"/>

<!--+
  | The fault elements.
  +-->
  <xsd:element name="InternalFault"            type="vos.types.
1.0:InternalFaultType"/>
  <xsd:element name="NodeNotFoundFault"       type="vos.types.
1.0:NodeNotFoundFaultType"/>
  <xsd:element name="DuplicateNodeFault"      type="vos.types.
1.0:DuplicateNodeFaultType"/>
  <xsd:element name="PermissionDeniedFault"   type="vos.types.
1.0:PermissionDeniedFaultType"/>
  <xsd:element name="InvalidUriFault"         type="vos.types.
1.0:InvalidUriFaultType"/>
  <xsd:element name="InvalidTokenFault"       type="vos.types.
1.0:InvalidTokenFaultType"/>
  <xsd:element name="InvalidArgumentFault"    type="vos.types.
1.0:InvalidArgumentFaultType"/>
  <xsd:element name="TypeNotSupportedFault"   type="vos.types.
1.0:TypeNotSupportedFaultType"/>
  <xsd:element name="ViewNotSupportedFault"   type="vos.types.
1.0:ViewNotSupportedFaultType"/>
  <xsd:element name="ProtocolNotSupportedFault" type="vos.types.
1.0:ProtocolNotSupportedFaultType"/>
  <xsd:element name="TransferFailedFault"     type="vos.types.
1.0:TransferFailedFaultType"/>
  <xsd:element name="InvalidDataFault"        type="vos.types.
1.0:InvalidDataFaultType"/>

</xsd:schema>

```

```
</wsdl:types>

<!--+
  | The GetViews messages.
  +-->
<wsdl:message name="GetViewsRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetViews"/>
</wsdl:message>
<wsdl:message name="GetViewsResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetViewsResponse"/>
</wsdl:message>

<!--+
  | The GetProtocols messages.
  +-->
<wsdl:message name="GetProtocolsRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetProtocols"/>
</wsdl:message>
<wsdl:message name="GetProtocolsResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetProtocolsResponse"/>
</wsdl:message>

<!--+
  | The GetProperties messages.
  +-->
<wsdl:message name="GetPropertiesRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetProperties"/>
</wsdl:message>
<wsdl:message name="GetPropertiesResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetPropertiesResponse"/>
</wsdl:message>

<!--+
  | The CreateNode messages.
  +-->
<wsdl:message name="CreateNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:CreateNode"/>
</wsdl:message>
<wsdl:message name="CreateNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:CreateNodeResponse"/>
</wsdl:message>
```

```
<!--+
  | The DeleteNode messages.
  +-->
<wsdl:message name="DeleteNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:DeleteNode"/>
</wsdl:message>
<wsdl:message name="DeleteNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:DeleteNodeResponse"/>
</wsdl:message>

<!--+
  | The MoveNode messages.
  +-->
<wsdl:message name="MoveNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:MoveNode"/>
</wsdl:message>
<wsdl:message name="MoveNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:MoveNodeResponse"/>
</wsdl:message>

<!--+
  | The CopyNode messages.
  +-->
<wsdl:message name="CopyNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:CopyNode"/>
</wsdl:message>
<wsdl:message name="CopyNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:CopyNodeResponse"/>
</wsdl:message>

<!--+
  | The GetNode messages.
  +-->
<wsdl:message name="GetNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetNode"/>
</wsdl:message>
<wsdl:message name="GetNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:GetNodeResponse"/>
</wsdl:message>
```

```
<!--+
  | The SetNode messages.
  +-->
<wsdl:message name="SetNodeRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:SetNode"/>
</wsdl:message>
<wsdl:message name="SetNodeResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:SetNodeResponse"/>
</wsdl:message>

<!--+
  | The ListNodes messages.
  +-->
<wsdl:message name="ListNodesRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:ListNodes"/>
</wsdl:message>
<wsdl:message name="ListNodesResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:ListNodesResponse"/>
</wsdl:message>

<!--+
  | The PushToVoSpace messages.
  +-->
<wsdl:message name="PushToVoSpaceRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PushToVoSpace"/>
</wsdl:message>
<wsdl:message name="PushToVoSpaceResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PushToVoSpaceResponse"/>
</wsdl:message>

<!--+
  | The PullToVoSpace messages.
  +-->
<wsdl:message name="PullToVoSpaceRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PullToVoSpace"/>
</wsdl:message>
<wsdl:message name="PullToVoSpaceResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PullToVoSpaceResponse"/>
</wsdl:message>

<!--+
```

```
| The PullFromVoSpace messages.
+-->
<wsdl:message name="PullFromVoSpaceRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PullFromVoSpace"/>
</wsdl:message>
<wsdl:message name="PullFromVoSpaceResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PullFromVoSpaceResponse"/>
</wsdl:message>

<!--+
| The PushFromVoSpace messages.
+-->
<wsdl:message name="PushFromVoSpaceRequestMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PushFromVoSpace"/>
</wsdl:message>
<wsdl:message name="PushFromVoSpaceResponseMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PushFromVoSpaceResponse"/>
</wsdl:message>

<!--+
| The fault messages.
+-->
<wsdl:message name="InternalFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:InternalFault"/>
</wsdl:message>
<wsdl:message name="NodeNotFoundFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:NodeNotFoundFault"/>
</wsdl:message>
<wsdl:message name="DuplicateNodeFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:DuplicateNodeFault"/>
</wsdl:message>
<wsdl:message name="PermissionDeniedFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:PermissionDeniedFault"/>
</wsdl:message>
<wsdl:message name="InvalidUriFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:InvalidUriFault"/>
</wsdl:message>
<wsdl:message name="InvalidTokenFaultMessage">
  <wsdl:part name="message" element="vos.contract.1.0:InvalidTokenFault"/>
</wsdl:message>
<wsdl:message name="InvalidArgumentFaultMessage">
```

```

    <wsdl:part name="message" element="vos.contract.1.0:InvalidArgumentFault"/>
</wsdl:message>
<wsdl:message name="TypeNotSupportedFaultMessage">
    <wsdl:part name="message" element="vos.contract.1.0:TypeNotSupportedFault"/>
</wsdl:message>
<wsdl:message name="ViewNotSupportedFaultMessage">
    <wsdl:part name="message" element="vos.contract.1.0:ViewNotSupportedFault"/>
</wsdl:message>
<wsdl:message name="ProtocolNotSupportedFaultMessage">
    <wsdl:part name="message" element="vos.contract.1.0:ProtocolNotSupportedFault"/>
</wsdl:message>
<wsdl:message name="TransferFailedFaultMessage">
    <wsdl:part name="message" element="vos.contract.1.0:TransferFailedFault"/>
</wsdl:message>
<wsdl:message name="InvalidDataFaultMessage">
    <wsdl:part name="message" element="vos.contract.1.0:InvalidDataFault"/>
</wsdl:message>

<!--+
| The VoSpace-1.0 port type.
+-->
<wsdl:portType name="VOspacePortType">

    <!--+
    | The GetViews operation.
    +-->
    <wsdl:operation name="GetViews">
        <wsdl:documentation>
            GetViews operation
        </wsdl:documentation>
        <wsdl:input message="vos.contract.1.0:GetViewsRequestMessage"/>
        <wsdl:output message="vos.contract.1.0:GetViewsResponseMessage"/>
        <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
    </wsdl:operation>

    <!--+
    | The GetProtocols operation.
    +-->
    <wsdl:operation name="GetProtocols">
        <wsdl:documentation>

```

```

        GetProtocols operation
    </wsdl:documentation>
    <wsdl:input  message="vos.contract.1.0:GetProtocolsRequestMessage"/>
    <wsdl:output message="vos.contract.1.0:GetProtocolsResponseMessage"/>
    <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
    </wsdl:operation>

<!--+
    | The GetProperties operation.
    +-->
    <wsdl:operation name="GetProperties">
        <wsdl:documentation>
            GetProperties operation
        </wsdl:documentation>
        <wsdl:input  message="vos.contract.1.0:GetPropertiesRequestMessage"/>
        <wsdl:output message="vos.contract.1.0:GetPropertiesResponseMessage"/>
        <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
    </wsdl:operation>

<!--+
    | The CreateNode operation.
    +-->
    <wsdl:operation name="CreateNode">
        <wsdl:documentation>
            CreateNode operation
        </wsdl:documentation>
        <wsdl:input  message="vos.contract.1.0:CreateNodeRequestMessage"/>
        <wsdl:output message="vos.contract.1.0:CreateNodeResponseMessage"/>
        <wsdl:fault name="InternalFault"          message="vos.contract.
1.0:InternalFaultMessage"/>
        <wsdl:fault name="InvalidUriFault"        message="vos.contract.
1.0:InvalidUriFaultMessage"/>
        <wsdl:fault name="DuplicateNodeFault"     message="vos.contract.
1.0:DuplicateNodeFaultMessage"/>
        <wsdl:fault name="TypeNotSupportedFault" message="vos.contract.
1.0:TypeNotSupportedFaultMessage"/>
        <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

<!--+

```

```

    | The DeleteNode operation.
    +-->
<wsdl:operation name="DeleteNode">
  <wsdl:documentation>
    DeleteNode operation
  </wsdl:documentation>
  <wsdl:input message="vos.contract.1.0:DeleteNodeRequestMessage"/>
  <wsdl:output message="vos.contract.1.0:DeleteNodeResponseMessage"/>
  <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
  <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
  <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
  <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
</wsdl:operation>

<!--+
    | The MoveNode operation.
    +-->
<wsdl:operation name="MoveNode">
  <wsdl:documentation>
    MoveNode operation
  </wsdl:documentation>
  <wsdl:input message="vos.contract.1.0:MoveNodeRequestMessage"/>
  <wsdl:output message="vos.contract.1.0:MoveNodeResponseMessage"/>
  <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
  <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
  <wsdl:fault name="InvalidArgumentFault" message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
  <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
  <wsdl:fault name="DuplicateNodeFault" message="vos.contract.
1.0:DuplicateNodeFaultMessage"/>
  <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
</wsdl:operation>

<!--+
    | The CopyNode operation.
    +-->

```

```

    <wsdl:operation name="CopyNode">
      <wsdl:documentation>
        CopyNode operation
      </wsdl:documentation>
      <wsdl:input message="vos.contract.1.0:CopyNodeRequestMessage"/>
      <wsdl:output message="vos.contract.1.0:CopyNodeResponseMessage"/>
      <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
      <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
      <wsdl:fault name="InvalidArgumentFault" message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
      <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
      <wsdl:fault name="DuplicateNodeFault" message="vos.contract.
1.0:DuplicateNodeFaultMessage"/>
      <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

    <!--+
      | The GetNode operation.
    +-->
    <wsdl:operation name="GetNode">
      <wsdl:documentation>
        GetNode operation
      </wsdl:documentation>
      <wsdl:input message="vos.contract.1.0:GetNodeRequestMessage"/>
      <wsdl:output message="vos.contract.1.0:GetNodeResponseMessage"/>
      <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
      <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
      <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
      <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

    <!--+
      | The SetNode operation.
    +-->
    <wsdl:operation name="SetNode">
      <wsdl:documentation>

```

```

        SetNode operation
    </wsdl:documentation>
    <wsdl:input message="vos.contract.1.0:SetNodeRequestMessage"/>
    <wsdl:output message="vos.contract.1.0:SetNodeResponseMessage"/>
    <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
    <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
    <wsdl:fault name="InvalidArgumentFault" message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
    <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
    <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

<!--+
    | The ListNodes operation.
    +-->
    <wsdl:operation name="ListNodes">
        <wsdl:documentation>
            ListNodes operation
        </wsdl:documentation>
        <wsdl:input message="vos.contract.1.0:ListNodesRequestMessage"/>
        <wsdl:output message="vos.contract.1.0:ListNodesResponseMessage"/>
        <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
        <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
        <wsdl:fault name="NodeNotFoundFault" message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
        <wsdl:fault name="InvalidTokenFault" message="vos.contract.
1.0:InvalidTokenFaultMessage"/>
        <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

<!--+
    | The PushToVoSpace operation.
    +-->
    <wsdl:operation name="PushToVoSpace">
        <wsdl:documentation>
            PushToVoSpace operation
        </wsdl:documentation>

```

```

        <wsdl:input message="vos.contract.1.0:PushToVoSpaceRequestMessage"/>
        <wsdl:output message="vos.contract.1.0:PushToVoSpaceResponseMessage"/>
        <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
        <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
        <wsdl:fault name="InvalidArgumentFault" message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
        <wsdl:fault name="ViewNotSupportedFault" message="vos.contract.
1.0:ViewNotSupportedFaultMessage"/>
        <wsdl:fault name="TypeNotSupportedFault" message="vos.contract.
1.0:TypeNotSupportedFaultMessage"/>
        <wsdl:fault name="ProtocolNotSupportedFault" message="vos.contract.
1.0:ProtocolNotSupportedFaultMessage"/>
        <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

<!--+
    | The PullToVoSpace operation.
    +-->
<wsdl:operation name="PullToVoSpace">
    <wsdl:documentation>
        PullToVoSpace operation
    </wsdl:documentation>
    <wsdl:input message="vos.contract.1.0:PullToVoSpaceRequestMessage"/>
    <wsdl:output message="vos.contract.1.0:PullToVoSpaceResponseMessage"/>
    <wsdl:fault name="InternalFault" message="vos.contract.
1.0:InternalFaultMessage"/>
    <wsdl:fault name="InvalidUriFault" message="vos.contract.
1.0:InvalidUriFaultMessage"/>
    <wsdl:fault name="InvalidDataFault" message="vos.contract.
1.0:InvalidDataFaultMessage"/>
    <wsdl:fault name="InvalidArgumentFault" message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
    <wsdl:fault name="ViewNotSupportedFault" message="vos.contract.
1.0:ViewNotSupportedFaultMessage"/>
    <wsdl:fault name="TypeNotSupportedFault" message="vos.contract.
1.0:TypeNotSupportedFaultMessage"/>
    <wsdl:fault name="ProtocolNotSupportedFault" message="vos.contract.
1.0:ProtocolNotSupportedFaultMessage"/>
    <wsdl:fault name="TransferFailedFault" message="vos.contract.
1.0:TransferFailedFaultMessage"/>
    <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
</wsdl:operation>

```

```

<!--+
  | The PullFromVoSpace operation.
  +-->
<wsdl:operation name="PullFromVoSpace">
  <wsdl:documentation>
    PullFromVoSpace operation
  </wsdl:documentation>
  <wsdl:input  message="vos.contract.1.0:PullFromVoSpaceRequestMessage"/>
  <wsdl:output message="vos.contract.1.0:PullFromVoSpaceResponseMessage"/>
  <wsdl:fault  name="InternalFault"           message="vos.contract.
1.0:InternalFaultMessage"/>
  <wsdl:fault  name="InvalidUriFault"        message="vos.contract.
1.0:InvalidUriFaultMessage"/>
  <wsdl:fault  name="InvalidArgumentFault"   message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
  <wsdl:fault  name="NodeNotFoundFault"      message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
  <wsdl:fault  name="ViewNotSupportedFault"  message="vos.contract.
1.0:ViewNotSupportedFaultMessage"/>
  <wsdl:fault  name="ProtocolNotSupportedFault" message="vos.contract.
1.0:ProtocolNotSupportedFaultMessage"/>
  <wsdl:fault  name="PermissionDeniedFault"  message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
</wsdl:operation>

<!--+
  | The PushFromVoSpace operation.
  +-->
<wsdl:operation name="PushFromVoSpace">
  <wsdl:documentation>
    PushFromVoSpace operation
  </wsdl:documentation>
  <wsdl:input  message="vos.contract.1.0:PushFromVoSpaceRequestMessage"/>
  <wsdl:output message="vos.contract.1.0:PushFromVoSpaceResponseMessage"/>
  <wsdl:fault  name="InternalFault"           message="vos.contract.
1.0:InternalFaultMessage"/>
  <wsdl:fault  name="InvalidUriFault"        message="vos.contract.
1.0:InvalidUriFaultMessage"/>
  <wsdl:fault  name="InvalidArgumentFault"   message="vos.contract.
1.0:InvalidArgumentFaultMessage"/>
  <wsdl:fault  name="NodeNotFoundFault"      message="vos.contract.
1.0:NodeNotFoundFaultMessage"/>
  <wsdl:fault  name="ViewNotSupportedFault"  message="vos.contract.
1.0:ViewNotSupportedFaultMessage"/>
  <wsdl:fault  name="ProtocolNotSupportedFault" message="vos.contract.
1.0:ProtocolNotSupportedFaultMessage"/>

```

```

        <wsdl:fault name="TransferFailedFault" message="vos.contract.
1.0:TransferFailedFaultMessage"/>
        <wsdl:fault name="PermissionDeniedFault" message="vos.contract.
1.0:PermissionDeniedFaultMessage"/>
    </wsdl:operation>

</wsdl:portType>

<!--+
    | The WebService HTTP binding.
+-->
<wsdl:binding name="VOspaceBinding" type="vos.contract.1.0:VOspacePortType">
    <wsdl:documentation>
        HTTP binding for the a VoSpace service.
    </wsdl:documentation>
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

<!--+
    | The GetViews operation.
+-->
<wsdl:operation name="GetViews">
    <wsdl:documentation>
        GetViews operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOspaceContract-
v1.0:GetViews"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
        <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
</wsdl:operation>

<!--+
    | The GetProtocols operation.
+-->
<wsdl:operation name="GetProtocols">

```

```

    <wsdl:documentation>
        GetProtocols operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:GetProtocols"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
        <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
</wsdl:operation>

<!--+
    | The GetProperties operation.
    +-->
<wsdl:operation name="GetProperties">
    <wsdl:documentation>
        GetProperties operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:GetProperties"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
        <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
</wsdl:operation>

<!--+
    | The CreateNode operation.
    +-->
<wsdl:operation name="CreateNode">
    <wsdl:documentation>

```

```

        CreateNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:CreateNode"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
        <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
        <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="DuplicateNodeFault">
        <soap:fault use="literal" name="DuplicateNodeFault"/>
    </wsdl:fault>
    <wsdl:fault name="TypeNotSupportedFault">
        <soap:fault use="literal" name="TypeNotSupportedFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
        <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
</wsdl:operation>

<!--+
| The DeleteNode operation.
+-->
<wsdl:operation name="DeleteNode">
    <wsdl:documentation>
        DeleteNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:DeleteNode"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>

```

```

    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

  <!--+
    | The MoveNode operation.
    +-->
  <wsdl:operation name="MoveNode">
    <wsdl:documentation>
      MoveNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:MoveNode"/>
    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidArgumentFault">
      <soap:fault use="literal" name="InvalidArgumentFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>

```

```

    </wsdl:fault>
    <wsdl:fault name="DuplicateNodeFault">
      <soap:fault use="literal" name="DuplicateNodeFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

  <!--+
    | The CopyNode operation.
  +-->
  <wsdl:operation name="CopyNode">
    <wsdl:documentation>
      CopyNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:CopyNode"/>
    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidArgumentFault">
      <soap:fault use="literal" name="InvalidArgumentFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>
    <wsdl:fault name="DuplicateNodeFault">
      <soap:fault use="literal" name="DuplicateNodeFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>

```

```

        </wsdl:fault>
    </wsdl:operation>

<!--+
    | The GetNode operation.
    +-->
<wsdl:operation name="GetNode">
    <wsdl:documentation>
        GetNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:GetNode"/>
    <wsdl:input>
        <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
        <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
        <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
        <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
        <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
</wsdl:operation>

<!--+
    | The SetNode operation.
    +-->
<wsdl:operation name="SetNode">
    <wsdl:documentation>
        SetNode operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:SetNode"/>

```

```

    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidArgumentFault">
      <soap:fault use="literal" name="InvalidArgumentFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

  <!--+
    | The ListNodes operation.
  +-->
  <wsdl:operation name="ListNodes">
    <wsdl:documentation>
      ListNodes operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOSpaceContract-
v1.0:ListNodes"/>
    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>

```

```

    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidTokenFault">
      <soap:fault use="literal" name="InvalidTokenFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

  <!--+
    | The PushToVoSpace operation.
    +-->
  <wsdl:operation name="PushToVoSpace">
    <wsdl:documentation>
      PushToVoSpace operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOspaceContract-
v1.0:PushToVoSpace"/>
    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidArgumentFault">
      <soap:fault use="literal" name="InvalidArgumentFault"/>
    </wsdl:fault>
    <wsdl:fault name="ViewNotSupportedFault">
      <soap:fault use="literal" name="ViewNotSupportedFault"/>
    </wsdl:fault>

```

```

    <wsdl:fault name="TypeNotSupportedFault">
      <soap:fault use="literal" name="TypeNotSupportedFault"/>
    </wsdl:fault>
  <wsdl:fault name="ProtocolNotSupportedFault">
    <soap:fault use="literal" name="ProtocolNotSupportedFault"/>
  </wsdl:fault>
  <wsdl:fault name="PermissionDeniedFault">
    <soap:fault use="literal" name="PermissionDeniedFault"/>
  </wsdl:fault>
</wsdl:operation>

<!--+
  | The PullToVoSpace operation.
  +-->
<wsdl:operation name="PullToVoSpace">
  <wsdl:documentation>
    PullToVoSpace operation.
  </wsdl:documentation>
  <soap:operation soapAction="http://www.net.ivoa/xml/VOspaceContract-
v1.0:PullToVoSpace"/>
  <wsdl:input>
    <soap:body use="literal" parts="message"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" parts="message"/>
  </wsdl:output>
  <wsdl:fault name="InternalFault">
    <soap:fault use="literal" name="InternalFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidUriFault">
    <soap:fault use="literal" name="InvalidUriFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidDataFault">
    <soap:fault use="literal" name="InvalidDataFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidArgumentFault">
    <soap:fault use="literal" name="InvalidArgumentFault"/>
  </wsdl:fault>
  <wsdl:fault name="ViewNotSupportedFault">
    <soap:fault use="literal" name="ViewNotSupportedFault"/>
  </wsdl:fault>

```

```

    <wsdl:fault name="TypeNotSupportedFault">
      <soap:fault use="literal" name="TypeNotSupportedFault"/>
    </wsdl:fault>
    <wsdl:fault name="ProtocolNotSupportedFault">
      <soap:fault use="literal" name="ProtocolNotSupportedFault"/>
    </wsdl:fault>
    <wsdl:fault name="TransferFailedFault">
      <soap:fault use="literal" name="TransferFailedFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

  <!--+
    | The PullFromVoSpace operation.
    +-->
  <wsdl:operation name="PullFromVoSpace">
    <wsdl:documentation>
      PullFromVoSpace operation.
    </wsdl:documentation>
    <soap:operation soapAction="http://www.net.ivoa/xml/VOspaceContract-
v1.0:PullFromVoSpace"/>
    <wsdl:input>
      <soap:body use="literal" parts="message"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" parts="message"/>
    </wsdl:output>
    <wsdl:fault name="InternalFault">
      <soap:fault use="literal" name="InternalFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidUriFault">
      <soap:fault use="literal" name="InvalidUriFault"/>
    </wsdl:fault>
    <wsdl:fault name="InvalidArgumentFault">
      <soap:fault use="literal" name="InvalidArgumentFault"/>
    </wsdl:fault>
    <wsdl:fault name="NodeNotFoundFault">
      <soap:fault use="literal" name="NodeNotFoundFault"/>
    </wsdl:fault>

```

```

    <wsdl:fault name="ViewNotSupportedFault">
      <soap:fault use="literal" name="ViewNotSupportedFault"/>
    </wsdl:fault>
  <wsdl:fault name="ProtocolNotSupportedFault">
    <soap:fault use="literal" name="ProtocolNotSupportedFault"/>
  </wsdl:fault>
  <wsdl:fault name="PermissionDeniedFault">
    <soap:fault use="literal" name="PermissionDeniedFault"/>
  </wsdl:fault>
</wsdl:operation>

<!--+
  | The PushFromVoSpace operation.
  +-->
<wsdl:operation name="PushFromVoSpace">
  <wsdl:documentation>
    PushFromVoSpace operation.
  </wsdl:documentation>
  <soap:operation soapAction="http://www.net.ivoa/xml/VOspaceContract-
v1.0:PushFromVoSpace"/>
  <wsdl:input>
    <soap:body use="literal" parts="message"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" parts="message"/>
  </wsdl:output>
  <wsdl:fault name="InternalFault">
    <soap:fault use="literal" name="InternalFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidUriFault">
    <soap:fault use="literal" name="InvalidUriFault"/>
  </wsdl:fault>
  <wsdl:fault name="InvalidArgumentFault">
    <soap:fault use="literal" name="InvalidArgumentFault"/>
  </wsdl:fault>
  <wsdl:fault name="NodeNotFoundFault">
    <soap:fault use="literal" name="NodeNotFoundFault"/>
  </wsdl:fault>
  <wsdl:fault name="ViewNotSupportedFault">
    <soap:fault use="literal" name="ViewNotSupportedFault"/>
  </wsdl:fault>

```

```

    <wsdl:fault name="ProtocolNotSupportedFault">
      <soap:fault use="literal" name="ProtocolNotSupportedFault"/>
    </wsdl:fault>
    <wsdl:fault name="TransferFailedFault">
      <soap:fault use="literal" name="TransferFailedFault"/>
    </wsdl:fault>
    <wsdl:fault name="PermissionDeniedFault">
      <soap:fault use="literal" name="PermissionDeniedFault"/>
    </wsdl:fault>
  </wsdl:operation>

```

```
</wsdl:binding>
```

```
<!--+
```

```
| The top level VoSpace-1.0 WebService definition.
```

```
+-->
```

```
<wsdl:service name="VOspaceService">
```

```
  <wsdl:documentation>...</wsdl:documentation>
```

```
  <wsdl:port name="VOspacePort" binding="vos.contract.1.0:VOspaceBinding">
```

```
    <soap:address location="..." />
```

```
  </wsdl:port>
```

```
</wsdl:service>
```

```
</wsdl:definitions>
```

6.2 Message schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:vos.types.1.0="http://www.net.ivoa/xml/VOspaceTypes-v1.0"
```

```
  targetNamespace="http://www.net.ivoa/xml/VOspaceTypes-v1.0"
```

```
  elementFormDefault="qualified"
```

```
>
```

```
<!-- ===== Property types ===== -->
```

```
<xsd:complexType name="PropertyType">
```

```
  <xsd:simpleContent>
```

```
    <xsd:extension base="xsd:string">
```

```
      <xsd:attributeGroup ref="vos.types.
```

```
1.0:PropertyAttributeGroup"></xsd:attributeGroup>
```

```

        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType >

<xsd:complexType name="PropertyListType">
    <xsd:annotation>
        <xsd:documentation>
            A container element for a list of properties.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="property" type="vos.types.1.0:PropertyType" minOccurs="0"
maxOccurs="unbounded" nillable="true"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PropertyReferenceType">
    <xsd:annotation>
        <xsd:documentation>
            A reference to a property description, used in getProperties()
        </xsd:documentation>
    </xsd:annotation>
    <xsd:attributeGroup ref="vos.types.
1.0:PropertyAttributeGroup"></xsd:attributeGroup>
</xsd:complexType>

<xsd:complexType name="PropertyReferenceListType">
    <xsd:annotation>
        <xsd:documentation>
            A container element for a list of property references.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="property" type="vos.types.1.0:PropertyReferenceType"
minOccurs="0" maxOccurs="unbounded" nillable="true"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetPropertyRequestType">
</xsd:complexType>

<xsd:complexType name="GetPropertyResponseType">

```

```

    <xsd:sequence>
      <xsd:element name="accepts" type="vos.types.1.0:PropertyReferenceListType">
        <xsd:annotation>
          <xsd:documentation>
            A list of identifiers for the properties that the service
accepts and understands.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="provides" type="vos.types.1.0:PropertyReferenceListType">
        <xsd:annotation>
          <xsd:documentation>
            A list of identifiers for the properties that the service
provides.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="contains" type="vos.types.1.0:PropertyReferenceListType">
        <xsd:annotation>
          <xsd:documentation>
            A list of identifiers for all the properties currently used by
nodes within the service.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

<!-- ===== View types ===== -->

<xsd:complexType name="ParamType">
  <xsd:annotation>
    <xsd:documentation>
      A view or protocol parameter.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="uri" type="xsd:anyURI" use="required"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

```

```

<xsd:complexType name="ViewType">
  <xsd:annotation>
    <xsd:documentation>
      An element describing a view of a data-set.
      A view may just provide the original data, or it could be server
generated.
      Examples of server generated views could include a votable view of data
in a database table,
      or a conversion from one image format to another.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="param" type="vos.types.1.0:ParamType" minOccurs="0"
maxOccurs="unbounded" nillable="true">
      <xsd:annotation>
        <xsd:documentation>
          A list of parameters for the view.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="uri" type="xsd:anyURI" use="required">
    <xsd:annotation>
      <xsd:documentation>
        The view URI.
        This should point to a resource describing the view format and what
parameters it requires.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="original" type="xsd:boolean" use="optional" default="true">
    <xsd:annotation>
      <xsd:documentation>
        A flag to indicate if the view provides access to the original data
content or a derived form.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="ViewListType">
  <xsd:annotation>

```

```

    <xsd:documentation>
        A container element for a list of views.
    </xsd:documentation>
</xsd:annotation>
<xsd:sequence>
    <xsd:element name="view" type="vos.types.1.0:ViewType" minOccurs="0"
maxOccurs="unbounded" nillable="true"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetViewsRequestType">
</xsd:complexType>

<xsd:complexType name="GetViewsResponseType">
    <xsd:sequence>
        <xsd:element name="accepts" type="vos.types.1.0:ViewListType">
            <xsd:annotation>
                <xsd:documentation>
                    A list of identifiers for the views that the service can accept.
                    A simple file based system may accept data in 'any' format.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
        <xsd:element name="provides" type="vos.types.1.0:ViewListType">
            <xsd:annotation>
                <xsd:documentation>
                    A list of identifiers for the views that the service can
provide.
                    A simple file based system may only provide data in the original
format.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

<!-- ===== Protocol types ===== -->

<xsd:complexType name="ProtocolType">
    <xsd:annotation>
        <xsd:documentation>
            A protocol element, containing the protocol URI, the endpoint and any

```

protocol specific parameters.

```

    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="endpoint" type="xsd:anyURI" minOccurs="0" maxOccurs="1">
      <xsd:annotation>
        <xsd:documentation>
          The target endpoint to use for a data transfer.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="param" type="vos.types.1.0:ParamType" minOccurs="0"
maxOccurs="unbounded" nillable="true">
      <xsd:annotation>
        <xsd:documentation>
          Any additional protocol specific parameters required to use the
endpoint.
          For example, the user name or password to use for ftp access.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="uri" type="xsd:anyURI" use="required">
    <xsd:annotation>
      <xsd:documentation>
        The protocol identifier.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="ProtocollistType">
  <xsd:annotation>
    <xsd:documentation>
      A container element for a list of protocols.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="protocol" type="vos.types.1.0:ProtocolType" minOccurs="0"
maxOccurs="unbounded" nillable="true"/>
  </xsd:sequence>
</xsd:complexType>

```

```
<xsd:complexType name="GetProtocolsRequestType">
</xsd:complexType>
```

```
<xsd:complexType name="GetProtocolsResponseType">
```

```
  <xsd:sequence>
```

```
    <xsd:element name="accepts" type="vos.types.1.0:ProtocolListType">
```

```
      <xsd:annotation>
```

```
        <xsd:documentation>
```

```
          A list of identifiers for the protocols that the service can
accept.
```

```
          This means that the service can act as a client for the
protocol.
```

```
        </xsd:documentation>
```

```
      </xsd:annotation>
```

```
    </xsd:element>
```

```
    <xsd:element name="provides" type="vos.types.1.0:ProtocolListType">
```

```
      <xsd:annotation>
```

```
        <xsd:documentation>
```

```
          A list of identifiers for the protocols that the service can
provide.
```

```
          This means that the service can act as a server for the
protocol.
```

```
        </xsd:documentation>
```

```
      </xsd:annotation>
```

```
    </xsd:element>
```

```
  </xsd:sequence>
```

```
</xsd:complexType>
```

```
<!-- ===== Node types ===== -->
```

```
<xsd:complexType name="NodeType">
```

```
  <xsd:annotation>
```

```
    <xsd:documentation>
```

```
      The base class for all nodes.
```

```
    </xsd:documentation>
```

```
  </xsd:annotation>
```

```
  <xsd:sequence>
```

```
    <xsd:element name="properties" type="vos.types.1.0:PropertyListType"
minOccurs="0" maxOccurs="1">
```

```
      <xsd:annotation>
```

```
        <xsd:documentation>
```

```
          The list of node properties.
```

```

        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="uri" type="xsd:anyURI" use="required">
    <xsd:annotation>
        <xsd:documentation>
            The node identifier URI.
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
</xsd:complexType>

<xsd:complexType name="DataNodeType">
    <xsd:annotation>
        <xsd:documentation>
            The base class for data nodes.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:NodeType">
            <xsd:sequence>
                <xsd:element name="accepts" type="vos.types.1.0:ViewListType"
minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation>
                            The list of views or data formats that this node can
accept.
                            A simple unstructured node may accept data in any
format.
                            A structured node may only accept data in specific
formats.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="provides" type="vos.types.1.0:ViewListType"
minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation>
                            The list of views or data formats that this node can
provide.
                            A simple unstructured node may only provide access to
the data in the original format.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

A structured node may provide different views of the data generated by the service.

```

        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="busy" type="xsd:boolean" use="optional"
default="false">
    <xsd:annotation>
        <xsd:documentation>
            A flag to indicate if the node content is available.
            This will be set to false while the data is being imported,
            or if the underlying service is busy.
        </xsd:documentation>
    </xsd:annotation>
</xsd:attribute>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="UnstructuredDataNodeType">
    <xsd:annotation>
        <xsd:documentation>
            An unstructured data node, containing unspecified content.
            The service does not need to understand or interpret the content.
            This type of node can accept any format, and only provides one view
returning the original data.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:DataNodeType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="StructuredDataNodeType">
    <xsd:annotation>
        <xsd:documentation>
            A structured data node, containing a specific data format that the
service has understands.
            This type of node may only accept specific data formats, and provide
different views of the
            data generated by the service.

```

```
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:DataNodeType">
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

<!-- ===== Fault types ===== -->

<xsd:complexType name="BaseFaultType">
    <xsd:annotation>
        <xsd:documentation>
            The base class for all faults.
        </xsd:documentation>
    </xsd:annotation>
</xsd:complexType>

<xsd:complexType name="ServiceFaultType">
    <xsd:annotation>
        <xsd:documentation>
            The base class for service faults.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:BaseFaultType">
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

<xsd:complexType name="InternalFaultType">
    <xsd:annotation>
        <xsd:documentation>
            Fault to indicate an internal error in the service.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:ServiceFaultType">
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
<xsd:complexType name="OperationFaultType">
  <xsd:annotation>
    <xsd:documentation>
      The base class for operation faults.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="vos.types.1.0:BaseFaultType">
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="NodeNotFoundFaultType">
  <xsd:annotation>
    <xsd:documentation>
      Fault to indicate that an expected node did not exist.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="vos.types.1.0:OperationFaultType">
      <xsd:sequence>
        <xsd:element name="uri" type="xsd:anyURI" nillable="true"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="DuplicateNodeFaultType">
  <xsd:annotation>
    <xsd:documentation>
      Fault to indicate that node already exists with a given URI.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="vos.types.1.0:OperationFaultType">
      <xsd:sequence>
        <xsd:element name="uri" type="xsd:anyURI" nillable="true"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="PermissionDeniedFaultType">
```

```
  <xsd:annotation>
```

```
    <xsd:documentation>
```

operation.
Fault to indicate that the user does not have permission to perform an

```
    </xsd:documentation>
```

```
  </xsd:annotation>
```

```
  <xsd:complexContent>
```

```
    <xsd:extension base="vos.types.1.0:OperationFaultType">
```

```
    </xsd:extension>
```

```
  </xsd:complexContent>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="InvalidUriFaultType">
```

```
  <xsd:annotation>
```

```
    <xsd:documentation>
```

Fault to indicate an invalid URI.

```
    </xsd:documentation>
```

```
  </xsd:annotation>
```

```
  <xsd:complexContent>
```

```
    <xsd:extension base="vos.types.1.0:OperationFaultType">
```

```
      <xsd:sequence>
```

```
        <xsd:element name="value" type="xsd:string" nillable="true"/>
```

```
      </xsd:sequence>
```

```
    </xsd:extension>
```

```
  </xsd:complexContent>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="InvalidTokenFaultType">
```

```
  <xsd:annotation>
```

```
    <xsd:documentation>
```

Fault to indicate an invalid continuation token.

```
    </xsd:documentation>
```

```
  </xsd:annotation>
```

```
  <xsd:complexContent>
```

```
    <xsd:extension base="vos.types.1.0:OperationFaultType">
```

```
      <xsd:sequence>
```

```
        <xsd:element name="value" type="xsd:string" nillable="true"/>
```

```
      </xsd:sequence>
```

```
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="InvalidArgumentFaultType">
    <xsd:annotation>
        <xsd:documentation>
            Fault to indicate an invalid argument for an operation.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:OperationFaultType">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TypeNotSupportedFaultType">
    <xsd:annotation>
        <xsd:documentation>
            Fault to indicate the service does not support the requested node type.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:OperationFaultType">
            <xsd:sequence>
                <xsd:element name="name" type="xsd:QName" nillable="true"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ViewNotSupportedFaultType">
    <xsd:annotation>
        <xsd:documentation>
            Fault to indicate the service does not support the requested view.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:OperationFaultType">
            <xsd:sequence>
                <xsd:element name="view" type="vos.types.1.0:ViewType"

```

```

nillable="true"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ProtocolErrorType">
  <xsd:annotation>
    <xsd:documentation>
      An extension to the protocol type to include an error message.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="vos.types.1.0:ProtocolType">
      <xsd:sequence>
        <xsd:element name="error" type="xsd:string" nillable="true"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ProtocolNotSupportedFaultType">
  <xsd:annotation>
    <xsd:documentation>
      Fault to indicate the service does not support any of the requested
protocols.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="vos.types.1.0:OperationFaultType">
      <xsd:sequence>
        <xsd:element name="protocol" type="vos.types.1.0:ProtocolErrorType"
minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="TransferFailedFaultType">
  <xsd:annotation>
    <xsd:documentation>

```

```

        Fault to indicate a data transfer has failed.
    </xsd:documentation>
</xsd:annotation>
<xsd:complexContent>
    <xsd:extension base="vos.types.1.0:OperationFaultType">
        <xsd:sequence>
            <xsd:element name="protocol" type="vos.types.1.0:ProtocolErrorType"
minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="InvalidDataFaultType">
    <xsd:annotation>
        <xsd:documentation>
            Fault to indicate an attempt to import invalid data into a Structured
node .
        </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="vos.types.1.0:OperationFaultType">
            <xsd:sequence>
                <xsd:element name="view" type="vos.types.1.0:ViewType"
nillable="true"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- ===== Node manipulation messages ===== -->

<xsd:complexType name="CreateNodeRequestType">
    <xsd:sequence>
        <xsd:element name="node" type="vos.types.1.0:NodeType">
            <xsd:annotation>
                <xsd:documentation>
                    A template for the new node.
                    This can include values for any of the user modifiable node
properties.
                    If the node identifier URI is set to 'vos://null', then the
service will generate
                    a new name for the node.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

```

The client cannot use this method to set the list of views accepted or provided by the node.

```

    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="CreateNodeResponseType">
  <xsd:sequence>
    <xsd:element name="node" type="vos.types.1.0:NodeType">
      <xsd:annotation>
        <xsd:documentation>
          Details of the new node.
          For extended node types this will be replaced by the full
element for the extended type,
          using xsi:type to indicate the node type.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="DeleteNodeRequestType">
  <xsd:sequence>
    <xsd:element name="target" type="xsd:anyURI">
      <xsd:annotation>
        <xsd:documentation>
          The target node identifier.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="DeleteNodeResponseType">
</xsd:complexType>

```

```

<xsd:complexType name="MoveNodeRequestType">
  <xsd:sequence>
    <xsd:element name="source" type="xsd:anyURI">

```

```

        <xsd:annotation>
            <xsd:documentation>
                The source node identifier.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="destination" type="vos.types.1.0:NodeType">
        <xsd:annotation>
            <xsd:documentation>
                A template for the new node.
                This can include values for any of the user modifiable node
properties.
                If the node identifier is set to 'vos://null', then the service
will generate
                a new name for the node.
                The client cannot use this method to set the list of views
accepted or provided by the node.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MoveNodeResponseType">
    <xsd:sequence>
        <xsd:element name="node" type="vos.types.1.0:NodeType">
            <xsd:annotation>
                <xsd:documentation>
                    Details of the new node.
                    For extended node types this will be replaced by the full
element for the extended type,
                    using xsi:type to indicate the node type.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CopyNodeRequestType">
    <xsd:sequence>
        <xsd:element name="source" type="xsd:anyURI">
            <xsd:annotation>

```

```

        <xsd:documentation>
            The source node identifier.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:element name="destination" type="vos.types.1.0:NodeType">
    <xsd:annotation>
        <xsd:documentation>
            A template for the new node.
            This can include values for any of the user modifiable node
properties.
            If the node identifier is set to 'vos://null', then the service
will generate
            a new name for the node.
            The client cannot use this method to set the list of views
accepted or provided by the node.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CopyNodeResponseType">
    <xsd:sequence>
        <xsd:element name="node" type="vos.types.1.0:NodeType">
            <xsd:annotation>
                <xsd:documentation>
                    Details of the new node.
                    For extended node types this will be replaced by the full
element for the extended type,
                    using xsi:type to indicate the node type.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="GetNodeRequestType">
    <xsd:sequence>
        <xsd:element name="target" type="xsd:anyURI">
            <xsd:annotation>
                <xsd:documentation>

```

The target node identifier.

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```
</xsd:element>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="GetNodeResponseType">
```

```
<xsd:sequence>
```

```
<xsd:element name="node" type="vos.types.1.0:NodeType">
```

```
<xsd:annotation>
```

```
<xsd:documentation>
```

Details of the node.

For extended node types this will be replaced by the full element for the extended type,

using xsi:type to indicate the node type.

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```
</xsd:element>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="SetNodeRequestType">
```

```
<xsd:sequence>
```

```
<xsd:element name="node" type="vos.types.1.0:NodeType">
```

```
<xsd:annotation>
```

```
<xsd:documentation>
```

A node element containing a list of properties to change with their new values.

A property with no value will empty the property.

To remove a property, set the xsi:nil attribute to true.

The client cannot use this method to modify the node type.

The client cannot use this method to modify the list of views accepted or provided by the node.

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```
</xsd:element>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="SetNodeResponseType">
```

```
<xsd:sequence>
```

```

    <xsd:element name="node" type="vos.types.1.0:NodeType">
      <xsd:annotation>
        <xsd:documentation>
          Updated details of the node.
          For extended node types this will be replaced by the full
element for the extended type,
          using xsi:type to indicate the node type.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<!-- ===== Node list messages ===== -->

<xsd:complexType name="NodeListType">
  <xsd:annotation>
    <xsd:documentation>
      A container element for the ListNodes request and response messages.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="token" type="xsd:string" minOccurs="0" maxOccurs="1">
      <xsd:annotation>
        <xsd:documentation>
          An optional continuation token from a previous request.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="limit" type="xsd:int" minOccurs="0" maxOccurs="1" >
      <xsd:annotation>
        <xsd:documentation>
          An optional limit indicating the maximum number of results in a
response.
          If no limit is specified, the service may return all the results
in one list,
          or it may impose its own internal limit.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name="detail" minOccurs="0" maxOccurs="1">
      <xsd:annotation>

```

```

    <xsd:documentation>
        An enumeration indicating the level of detail required.
    </xsd:documentation>
</xsd:annotation>
<xsd:simpleType>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="min">
            <xsd:annotation>
                <xsd:documentation>
                    The minimum level of detail, returning simple node
elements with no child elements.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="properties">
            <xsd:annotation>
                <xsd:documentation>
                    An intermediate level of detail, returning basic
node elements including
                    the node properties.
                    Note - this hides the underlying node type.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="max">
            <xsd:annotation>
                <xsd:documentation>
                    The maximum level of detail, returning complex node
elements, including the
                    full details of any extended node types.
                    This uses the xsi:type attribute to include the type
specific elements of any
                    extended node types.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="nodes" minOccurs="0" maxOccurs="1">
    <xsd:annotation>
        <xsd:documentation>

```

The list of nodes.

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```
<xsd:complexType>
```

```
<xsd:sequence>
```

```
<xsd:element name="node" type="vos.types.1.0:NodeType"
minOccurs="0" maxOccurs="unbounded">
```

```
<xsd:annotation>
```

```
<xsd:documentation>
```

by the full element

```
At the maximum level of detail this will be replaced
```

the node type.

```
for the extended type, using xsi:type to indicate
```

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```
</xsd:element>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
</xsd:element>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="ListNodesRequestType">
```

```
<xsd:sequence>
```

```
<xsd:element name="request" type="vos.types.1.0:NodeListType"/>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<xsd:complexType name="ListNodesResponseType">
```

```
<xsd:sequence>
```

```
<xsd:element name="response" type="vos.types.1.0:NodeListType"/>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
<!-- ===== Transfer types ===== -->
```

```
<xsd:complexType name="TransferType">
```

```
<xsd:annotation>
```

```
<xsd:documentation>
```

```
A container element for transfer information.
```

```
</xsd:documentation>
```

```
</xsd:annotation>
```

```

    <xsd:sequence>
      <xsd:element name="view" type="vos.types.1.0:ViewType" minOccurs="0"
maxOccurs="1">
        <xsd:annotation>
          <xsd:documentation>
            For an import message, this indicates the data type being sent.
            For an export message, this indicates which view to request the
data from.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="protocol" type="vos.types.1.0:ProtocolType" minOccurs="0"
maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            A list of protocols to use for the transfer.
            In a request, this should contain a list of the protocols that
the client can use.
            In a response, this should contain the subset of requested
protocols that the service can support.
            The service should fill in the endpoint and parameters for each
protocol.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

<!-- ===== Data transfer messages ===== -->

<xsd:complexType name="PushToVoSpaceRequestType">
  <xsd:sequence>
    <xsd:element name="destination" type="vos.types.1.0:NodeType"/>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PushToVoSpaceResponseType">
  <xsd:sequence>
    <xsd:element name="destination" type="vos.types.1.0:NodeType"/>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

```

```
<xsd:complexType name="PullToVoSpaceRequestType">
  <xsd:sequence>
    <xsd:element name="destination" type="vos.types.1.0:NodeType"/>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PullToVoSpaceResponseType">
  <xsd:sequence>
    <xsd:element name="destination" type="vos.types.1.0:NodeType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PullFromVoSpaceRequestType">
  <xsd:sequence>
    <xsd:element name="source" type="xsd:anyURI"/>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PullFromVoSpaceResponseType">
  <xsd:sequence>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PushFromVoSpaceRequestType">
  <xsd:sequence>
    <xsd:element name="source" type="xsd:anyURI"/>
    <xsd:element name="transfer" type="vos.types.1.0:TransferType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PushFromVoSpaceResponseType">
</xsd:complexType>

<xsd:attributeGroup name="PropertyAttributeGroup">
  <xsd:attribute name="uri" type="xsd:anyURI" use="required">
    <xsd:annotation>
      <xsd:documentation>
```

If the property has been registered, then the URI should point to the registration document.

Third party tools may use the urn:xxxx syntax to add unregistered properties.

```

    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="readonly" type="xsd:boolean" use="optional"
default="false">
  <xsd:annotation>
    <xsd:documentation>
      A flag to indicate if the property is considered read-only.
      Attempting to modify a read-only property should generate a
PermissionDenied fault.
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
</xsd:attributeGroup>
</xsd:schema>
```

7 References

- [1] T. Berners-Lee, R. Fielding, U. Irvine, L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax, <http://www.faqs.org/rfcs/rfc2396.html>
- [2] R. Plante, T. Linde, R. Williams, & K. Noddle, IVOA Identifiers, <http://www.ivoa.net/Documents/latest/IDs.html>