DAP Web Services Specification DRAFT

James Gallagher[*], Nathan Potter[†], John Chamberlin

Printed: February 6, 2004
Revision: 1.5

1 Introduction

The web approach to building distributed applications has been more rapidly and widely adopted than any other approach. A logical extension of this success is the development of web services technologies which are more loosely coupled than traditional distributed programming models like RPC, DCOM and CORBA. Web services create interactions between clients and servers which are simple and flexible. At the same time the technology includes the structural capability to do object transfers that are not possible under plain HTTP. Web services do these transfers by using SOAP messages (as opposed to the MIME messages typical of ordinary web applications). This is a more data-friendly mechanism for exchanging information over the web and is ideal for the purpose of creating a convenient interface to OPeNDAP servers.

The rationale for creating OPeNDAP web services is that it would allow programmatic clients to access OPeNDAP servers in a more reliable and structured way than the HTTP interface. Web services also make client implementations easier because the client developer need only work with objects rather than worry about constructing and parsing text messages. Traditionally all of the control structures in OPeNDAP have been passed as text which both client and server must construct/parse. In the past we have provided tools such as the dods.dap Java package which allow the client to parse these messages, but this is a non-standard and incomplete solution. With web services OPeNDAP has a standard mechanism for reliable interfacing and object transfer without parsing unique grammars.

An important benefit of a web services interface is that it provides the basis for directory services via UDDI. Using this technology server sites may have the option of publishing information about their content globally or locally. This would expand the ability of end users to locate data sources.

Another auxiliary capability of the web services implementation will be to provide for the possibility of hosting secure sites–something not possible with current incarnations of OPeNDAP. For those users with sensitive or restricted data this would allow them to publish that data over the web, but still enforce a comprehensive security policy. SOAP supports an integrated ability to exchange digital signatures and X.509 credentials in a standard way.

Web services in OPeNDAP are divided into two major classes, Foundation Services and SEXYNAMEGOESHERE Services.

2 Foundation Services

The core services in the OPeNDAP hierarchy are called the *Foundation Services*.

There are 4 of these services, **GetDDX**, **GetData**, **GetBlobData** and **GetBlob**. The **GetDDX**, **GetData**, and **GetBlobData** services are based on SOAP messaging (as opposed to SOAP RPC). The **GetBlob** service does not use SOAP: It provides access to a serialized binary encoded data stream in an out-of-band manner.

[*]The University of Rhode Island, jgallagher@gso.uri.edu
[†]Oregon State University, ndp@coas.oregonstate.edu

The *Foundation Services* are all exposed through the single service name: *OPeNDAP*

This was done in order to allow clients to pool requests. A client may send a single message to the *OPeNDAP* service that contains multiple requests (see Section 2.6 on page 6).

NOTE: More Thoughts On Services
I think that for the sake of simplicity with respect to client software, complexity of servers, and especially with regards to long term configuration stability we should use a single foundation service terminus which is based on SOAP messaging. As higher level services (directory, catalog, and discovery) are added they should be implemented using SOAP RPC.
This allows clients to use the same messaging scheme, whether it's talking directly to a core server (listening on a TCP/IP socket or whathave you) or to a SOAP service. – ndp 2/7/04

2.1 Ethan Davis Had this comment:

I like the single 'OPeNDAP' service idea with the requested foundation service encoded in the request message. I wonder if the DDX should contain a more generic request message rather than one specific for the Blob, e.g.,

```
</Dataset>
    ...
    <RequestMsg SrvcURL="http://hostname/axis/services/OPeNDAP"
                dataSetName="data.set.name">
            <Constraint>
                ...
            </Constraint>
    </RequestMsg>
</Dataset>
```

That way it doesn't limit the type of request a user can build from the DDX info (or rather, it doesn't give the appearance of limiting the users request).

– Main.EthanDavis - 31 Jan 2004

2.2 GetDDX Service

The **GetDDX** service is used to provide client access to the DDX representation of a *Dataset* (see DAP Objects paper). The client must send a SOAP message containing a valid constraint expression to the service. In response the service will return the DDX of the *Dataset*, constrained as indicated in the constraint expression. This DDX will contain the message and service URL for the **GetBlob** service call that will return the Blob data. This service is the functional equivalent of the old DODS URL .dds response.

For example, the SOAP body of a **GetDDX** message might look like:

```
    <soap:Body>
        <GetDDX name="data.set.name">
            <Constraint>
                .
                .
                .
            </Constraint>
        </GetDDX>
    </soap:Body>
```

The SOAP body of the returned message might look like:

```
<soap:Body>
    <Dataset name="data.set.name">
        <Array name="sst">
            .
            .
            .
        </Array>
        <BlobRequestMsg SrvcURL="hostname:12001/OPeNDAP/BSrvc">
            <GetBlob name="data.set.name">
                <Constraint>
                    .
                    .
                    .
                </Constraint>
            </GetBlob>
        </GetBlobRequestMsg>
    </Dataset>
</soap:Body>
```

2.3 GetData Service

The **GetData** service is used to provide client access (via SOAP) to the DDX representation of a *Dataset* (see DAP Objects paper) bundled with the Blob that contains the serialized binary data content of the returned DDX. The client must send a SOAP message containing a valid constraint expression to the service. In response the service will return the DDX of the *Dataset*, constrained as indicated in the constraint expression. This will be followed (in the same message) by a *¡BlobData¿ element* whose content is the Base64 encoded serialized binary data content of the returned DDX constrained as specified in the constraint expression. As always, the DDX will contain the both the service URL for the **GetBlob** service call and the message to send to the service URL that will return the Blob data. This service is the functional equivalent of the old DODS URL .dods response.

An example call to the **GetDDX** service might look like:

```
<soap:Body>
    <GetData name="data.set.name">
        <Constraint>
            .
            .
            .
        </Constraint>
    </GetData >
</soap:Body>
```

The SOAP body of the returned message might look (with liberties taken with the XML formatting for readability) something like:

```
<soap:Body>
    <Dataset name="data.set.name">
        <Array name="sst">
            .
            .
            .
        </Array>
        <BlobRequestMsg SrvcURL="hostname:12001/OPeNDAP/BSrvc">
            <GetBlob name="data.set.name">
                <Constraint>
                    .
                    .
                    .
                </Constraint>
            </GetBlob>
        </GetBlobRequestMsg>
    </Dataset>
    <BlobData>
    7xUzYGtGWgAAABkAAAAkVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgMCk
    uAAAAJFRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDEpLgAAACRUaGlzIGl
    zIGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyAyKS4AAAAkVGhpcyBpcyBhIGRhdGEgdGV
    zdCBzdHJpbmcgKHBhc3MgMykuAAAAJFRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nICh
    wYXNzIDQpLgAAACRUaGlzIGlzIGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyA1KS4AAAA
    kVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgNikuAAAAJFRoaXMgaXMgYSB
    kYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDcpLgAAACRUaGlzIGlzIGEgZGF0YSB0ZXN0IHN
    0cmluZyAocGFzcyA4KS4AAAAkVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3M
    gOSkuAAAAJVRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDEwKS4AAAAAAAA
    lVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgMTEpLgAAAAAACVUaGlzIGl
    IGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyAxMikuAAAAAAAAJVRoaXMgaXMgYSBkYXRh
    </BlobData>

</soap:Body>
```

Although **GetData** service provides a full SOAP based interface to data values stored in an OPeNDAP server, it is strongly recommend that clients combine the use of the **GetDDX** service and the **GetBlob** service instead. Since the **GetData** service uses SOAP to carry binary data content the Base64 encoding of the data is necessary to allow the data to be transmitted in an XML document. This has (at least) two unfortunate ramifications: First, the number of bytes transmitted is increased by a factor of 33 percent. (CHECK THIS CLAIM!) Secondly, the entire response must be sent in the document, which eliminates the possibility of streaming the response to the client so that the client can process it on the fly.

2.4 GetBlobData Service

The **GetBlobData** service is used to provide client access (via SOAP) to the Blob that contains the serialized binary data content of the *Dataset* as detailed in the request message. The client must send a SOAP message containing a valid constraint expression to the service. In response the service will return a *¡BlobData¿ element* whose content is the Base64 encoded serialized binary data content of the returned DDX constrained as specified in the constraint expression.

This service is the functional equivalent of the old DODS URL .blob response, with the caveat that the binary data has been Base64 encoded.

```
<soap:Body>
    <GetBlobData name="data.set.name">
        <Constraint>
            .
            .
            .
        </Constraint>
    </GetBlobData >
</soap:Body>
```

The SOAP body of the returned message might look (with liberties taken with the XML formatting for readability) something like:

```
<soap:Body>
    <BlobData name="data.set.name">
    7xUzYGtGWgAAABkAAAAkVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgMCk
    uAAAAJFRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDEpLgAAACRUaGlzIGl
    zIGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyAyKS4AAAAkVGhpcyBpcyBhIGRhdGEgdGV
    zdCBzdHJpbmcgKHBhc3MgMykuAAAAJFRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nICh
    wYXNzIDQpLgAAACRUaGlzIGlzIGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyA1KS4AAAA
    kVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgNikuAAAAJFRoaXMgaXMgYSB
    kYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDcpLgAAACRUaGlzIGlzIGEgZGF0YSB0ZXN0IHN
    0cmluZyAocGFzcyA4KS4AAAAkVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3M
    gOSkuAAAAJVRoaXMgaXMgYSBkYXRhIHRlc3Qgc3RyaW5nIChwYXNzIDEwKS4AAAAAAAA
    lVGhpcyBpcyBhIGRhdGEgdGVzdCBzdHJpbmcgKHBhc3MgMTEpLgAAAAAAACVUaGlzIGl
    IGEgZGF0YSB0ZXN0IHN0cmluZyAocGFzcyAxMikuAAAAAAAAJVRoaXMgaXMgYSBkYXRh
    </BlobData>

</soap:Body>
```

Although **GetBlobData** service provides a full SOAP based interface to data values stored in an OPeNDAP server, it is strongly recommend that clients combine the use of the **GetDDX** service and the **GetBlob** service instead. Since the **GetBlobData** service uses SOAP to carry the binary data content the Base64 encoding of the serialized binary data is necessary to allow the data to be transmitted in an XML document. This has (at least) two unfortunate ramifications: First, the number of bytes transmitted is increased by a factor of 33 percent. (CHECK THIS CLAIM!) Secondly, the entire response must be sent in the document, which eliminates the possibility of streaming the response to the client so that the client can process it on the fly.

2.5 GetBlob Service

The **GetBlob** service is NOT a SOAP interface. This foundation service is used by clients to retrieve the serialized binary content of a *Dataset*. To invoke this service a client connects to the correct port on an OPeNDAP server and sends an XML message to the server. The OPeNDAP server replies with the serialized binary data content of the *Dataset* described in the request as described in the DAP Objects paper.

NOTE: The Blob return is actually more complicated than a simple stream. It must be tagged and chunked in such a way that if an error occurs the server can inject the error into the blob stream and the client can get the error and handle it in some cogent manner. At the moment this whole description lives, I think, in the DAP Objects paper but it may live only in the brain of James Gallagher... – ndp 2/7/04

An example **GetBlob** message might look like:

```
    <GetBlob name="data.set.name">
        <Constraint>
            .
            .
            .
        </Constraint>
    </GetBlob >
```

The server will return ”THE STUFF” (see the Note above) in response.

2.6 Pooled Requests

The *OPeNDAP* service supports pooling requests. A client may collect multiple **GetDDX**, **GetData**, and **GetBlobData** service requests in a single call to the server. The server will return the appropriate items in the same order that they were requested. All of the SOAP based *Foundation Services* may be pooled. The **GetBlob** service calls MAY NOT be pooled. (**At least I don't think so... I don't see how the raw inary serialization would support it.**)

For example, a client could request 3 DDX's in a single pooled request:

```
    <soap:Body>
        <GetDDX name="data.set.01">
            <Constraint/>
        </GetDDX>
        <GetDDX name="data.set.02">
            <Constraint/>
        </GetDDX>
        <GetDDX name="data.set.03">
            <Constraint/>
        </GetDDX>
    </soap:Body>
```

And the server would return:

```
    <soap:Body>
        <Dataset name="data.set.01">
            .
            .
            .
        </Dataset>
        <Dataset name="data.set.02">
            .
            .
            .
        </Dataset>
        <Dataset name="data.set.03">
            .
            .
            .
        </Dataset>
    </soap:Body>
```

The pooled requests need not be homogeneous. Consider:

```
<soap:Body>
    <GetDDX name="data.set.01">
        <Constraint/>
    </GetDDX>
    <GetData name="data.set.02">
        <Constraint/>
    </GetDDX>
    <GetDDX name="data.set.03">
        <Constraint/>
    </GetDDX>
    <GetBlobData name="data.set.04">
        <Constraint/>
    </GetBlobData >
</soap:Body>
```

And the server would return:

```
<soap:Body>
    <Dataset name="data.set.01">
        .
        .
        .
    </Dataset>
    <Dataset name="data.set.02">
        .
        .
        .
    </Dataset>
    <BlobData>
        .
        .
        .
    </BlobData>
    <Dataset name="data.set.03">
        .
        .
        .
    </Dataset>
    <BlobData name="data.set.04">
        .
        .
        .
    </BlobData>

</soap:Body>
```

3 SEXYNAMEGOESHERE Services

The SEXYNAMEGOESHERE services, such as directories and catalogues, are implemented them using SOAP RPC. These higher level services provide the kind of functionality that will allow our users to use UDDI as a discovery mechanism, and should be kept in the SOAP RPC domain.

3.1 GetDir Service

The **GetDir** service uses SOAP RPC. It is a parameterless call that returns a SOAP array of Strings containing the names of all the data sets at this server (probably at the granule level). These names are NOT URL/URI based, but simply the name that would be used in a CE passed to the GetDDX service.

3.2 GetInfo Service

The **GetInfo** service uses SOAP RPC. It takes a single String as a parameter; the name of the *Dataset* about which information is required. **GetInfo** returns an HTML document the when rendered provides a human readable page of information about the *Dataset* named in the passed parameter.

Errors: NoSuchDataset.

4 WSDL

5 UDDI

A Web Services and SOAP

SOAP is used in two basic ways, as a tool for doing RPC on remote systems, and as a messaging service.

When SOAP is used to perform RPC the parameters passed and the return values must be consistent with the SOAP data model (unless custom encodings are used, more on that later). This model is relatively simplistic, it contains a collection of atomic types, arrays, and structures. In the SOAP data model values must be associated with each instance of an atomic type, this includes members of an array or structure.

Given the type constraints and the embbeded value requirements of the SOAP data model it is not practical to try to shoe-horn the OPeNDAP data model directly into a SOAP data model representation model.

In order pass our DAP objects via SOAP methods we have 2 choices: Extend the SOAP data model to cover our custom data types (so that our types can be used in an RPC request) or rely on SOAP messaging.

Custom data types require the use of custom serialization and de-serialization classes that must be integrated into the SOAP server (and client). These classes are tightly coupled to the particular implementation of the SOAP engine (such as Apache AXIs, GLUE, etc.)

Messaging allows us to pass W3C DOM trees back and forth between server and client. Each side can then act upon them as necessary (parse them into a java/C++ class in memory, extract pertinent information directly from the DOM elements, what have you)

At the time of this writing it appears to me that using a SOAP messaging scheme may cut us off from the automatic WSDL generation tools that have been developed for the SOAP RPC model. However, this may not be such a large issue. Ultimately, having a custom data type in an RPC call only buys you the information (at the WSDL level) that this method requires (for example) a ”DDS” or a ”ConstraintExpression” but probably doesn’t lead you to an implementation of such a beast.