Neil P. Chue Hong, OMII-UK (editor)
Michel Drescher, Fujitsu Laboratories of Europe
Amy Krause, EPCC, University of Edinburgh
M. Shahbaz Memon, Forschungszentrum Jülich
Mark Morgan, University of Virginia
October 8, 2008

OGSA® ByteIO Implementations – Experiences Document

Status of This Document

Copyright Notice

Trademark

Abstract

This document reports about experiences made with running OGSA ByteIO working group interoperability test scenarios based on the ByteIO OGSA® WSRF Basic Profile Rendering 1.0 of the ByteIO Specification 1.0 as implemented by the four groups participating in the OGSA ByteIO Interoperability Fiesta. The four groups represented different implementation environments: Genesis II by the University of Virginia (UVa), UNICORE by Forschungszentrum Jülich, OGSA-DAI by EPCC, the University of Edinburgh (EPCC), and a clean-room implementation by Fujitsu Labs of Europe (FLE).
This document also remarks on the special considerations made to implement the OGSA ByteIO specifications on different Web Services and XML stacks. The main difficulties arise because of the way that different Web Services and XML tooling interprets particular elements. These arose not in the implementation of the ByteIO specification itself, but in the associated implementation required for the interoperability experiments.

Neil P. Chue Hong, OMII-UK (editor)
Michel Drescher, Fujitsu Laboratories of Europe
Amy Krause, EPCC, University of Edinburgh
M. Shahbaz Memon, Forschungszentrum Jülich
Mark Morgan, University of Virginia
October 8, 2008

Contents

Abstract 1
1. Introduction 3
2. Participants 3
a. Forschungszentrum Jülich (UNICORE) 3
b. EPCC, The University of Edinburgh (OGSA-DAI) 4
c. UVa (Genesis II) 4
d. FLE (clean room prototype) 5
3. Interoperability testing setup 5
a. Process and Infrastructure 5
4. Interoperability testing results 5
a. General Status of the Services 5
b. Interoperability Tables 6
Section 4.1 [RByteIO GetResourceProperty] 6
Section 4.2 [RByteIO Read i] 6
Section 4.3 [RByteIO Read ii] 6
Section 4.4 [RByteIO Read iii] 6
Section 4.5 [RByteIO Write i] 6
Section 4.6 [RByteIO Write ii] 7
Section 4.7 [RByteIO Write iii] 7
Section 4.8 [RByteIO Append] 7
Section 4.9 [RByteIO TruncAppend] 7
Section 5.1 [SByteIO GetResourceProperty] 7
Section 5.2 [SByteIO SeekRead] 7
Section 5.3 [SByteIO SeekWrite] 7
Missing Tags in SOAP Body of Request Message 8
Incorrect Resource Property Name 8
Ambiguity: Queries and Namespaces 9
Missing Tag in SOAP Body of Response Message 9
Missing Tags in SOAP Body of Request Message 9
Ambiguity: Queries and Namespaces 9
Missing Tag in SOAP body of Response Message 10
Incorrect Write Value of *Transfer-Information* 10
Wrong Treatment of Transfer-Information-Type XML element 11
Occurs throughout document 11
ByteIO Interop Port Type, DeleteResource, and Race Conditions 13
Incorrect Blocks-per-Byte in SOAP Example 14
5. Discussion 15
a. FZJ specific discussion 15
b. EPCC specific discussion 15
c. UVa specific discussion 15
d. FLE specific discussion 16
6. Conclusion 16
7. Security Considerations 17
8. Contributors 17
9. Intellectual Property Statement 18
10. Disclaimer 18
11. Full Copyright Notice 18
12. References 19

1. Introduction

The ByteIO Specification 1.0 document (GFD-R-P.087) [MORGAN1] specifies a set of port types that give users a concise, standard way of interacting with bulk data sources and sinks in the grid. The purpose of these port types is to provide the means for treating such data resources as POSIX-like files. ByteIO is divided into two port types and each addresses a unique set of use cases. The first of these port types supports the notion that a data resource is directly accessible and that clients can handle the maintenance of any session state (such as file pointer, buffering, caching, etc.). The other port type presents a more stream-like interface to clients and as such contains implicit session state. In this latter case data resources with this port type don't represent that bulk data source/sink directly but rather represent the resource of the open stream between the client and the data source/sink.

Soon after the ByteIO Specification reached "proposed recommendation" status, various Grid community oriented open source projects started implementing ByteIO endpoints. This document reports the experiences from running OGSA ByteIO working group interoperability test scenarios based on the ByteIO OGSA WSRF Basic Profile Rendering 1.0 (GFD-R-P.088) [MORGAN2] of the ByteIO Specification 1.0 as implemented by the four groups participating in the OGSA ByteIO Interoperability Fiesta. The Interoperability Test Scenarios are defined in OGSA-ByteIO Interoperability Testing Specification [DRESCHER], currently a draft Informational document going through the OGF document process.

2. Participants

Four groups participated in the ByteIO Interoperability Fiesta, which was held from May 7, 2007 to July 31, 2007. The four groups represented different implementation environments: Genesis II[1] by the University of Virginia (UVa), UNICORE[2] by Forschungszentrum Jülich, OGSA-DAI[3] by EPCC, the University of Edinburgh (EPCC), and a clean-room implementation by Fujitsu Labs of Europe (FLE).

In this section, each group describes the environment in which they implemented the ByteIO specification, and their motivating scenario for their use of ByteIO.

a. Forschungszentrum Jülich (UNICORE)

Forschungszentrum Jülich (FZJ) participated in the OGSA-ByteIO Interoperability fiesta by presenting a UNICORE file transfer service implementation. UNICORE provides intuitive and seamless access to computing resources across the web. It leverages the ByteIO specification for the file staging purposes along with other core services.

The ByteIO specification has been implemented as a web service interface with WSRF rendering. It uses UNICORE's WSRF hosting environment: an XFire[4] hosting environment with additional WSRF implementation elements. In order to handle XML bindings, it uses the Apache XMLBeans[5] helper library to handle the serialization and deserialization of XML message data types. The rationale behind the use of XMLBeans is its almost complete support of the XML-Schema implementation and also its provision of an interface to query instance documents.

[1] http://vcgr.cs.virginia.edu/genesisII/
[2] http://www.unicore.eu/
[3] http://www.ogsadai.org.uk/
[4] http://xfire.codehaus.org/
[5] http://xmlbeans.apache.org/

UNICORE's WSRF hosting environment implements the WS-Addressing specification [version 1.0; namespace "http://www.w3.org/2005/08/addressing"] to represent the addressing of web services. The hosting environment uses reference parameters to represent WS-endpoints exposed as WS-Resources. Therefore, Random and Streamable-ByteIO are exposed as WS-Resources, and therefore addressed via WS-Addressing's reference parameters.

UNICORE supports both the Random and Streamable interfaces of the ByteIO specification. UNICORE takes security as a prime concern for all of its services; therefore, ByteIO services allow secure access to file transfer interfaces. FZJ has implemented all mandatory and optional elements of the ByteIO specification with the exception of RandomByteIO “Last Access Time”, which has been omitted for the same reasons as described in the FLE implementation notes.

b. EPCC, The University of Edinburgh (OGSA-DAI)

EPCC participated in the interoperability testing with a prototype implementation of the RandomByteIO interfaces for OGSA-DAI. OGSA-DAI is a data access and integration framework, implemented in Java, which supports the exposure of data resources, such as relational or XML databases, on to grids. It uses Apache Axis 1.4[6] for handling SOAP messages and the Apache WS-Addressing implementation.

OGSA-DAI provides an implementation of WS-RF resources and resource properties which was extended for ByteIO resources. Since RandomByteIO resources are both OGSA-DAI resources and ByteIO resources they can be accessed from OGSA-DAI service interfaces as well as through the ByteIO interface implementation.

An example of a typical scenario that is envisaged for OGSA-DAI and ByteIO is the following: Binary data is stored in a database. This data is retrieved using OGSA-DAI data access functionality and written to a ByteIO resource. The RandomByteIO interface provides random access to the data in the resource.

c. UVa (Genesis II)

The Genesis II project at the University of Virginia (UVa) participated in the OGSA-ByteIO Interoperability testing by standing up an unsecured instance of its Genesis II grid. No modifications were made to the standard Genesis II deployment aside from the suppression of normal security mechanisms and the addition of the requisite factory service as indicated by the OGSA ByteIO Interoperability Fiesta document.

Genesis II uses Jetty as a front end to manage HTTP connections, Apache Axis 1.4 to multiplex SOAP messages out to appropriate web services implementation classes, WSS4J to handle message level security, and Apache Derby as a default backend database for persistence. Further, the Genesis II project includes a home-grown implementation of the WS-Addressing specification (corresponding to WS-Addressing namespace http://www.w3.org/2005/08/addressing). Additional ancillary libraries are used for edge-case service implementations.

The Genesis II project is founded off of the belief that using familiar user interaction abstractions increases the adoptability of the grid and to this end makes heavy use of ByteIO and RNS together to provide the illusion that users are interacting with a file system when in reality they are interacting with the grid. Thus, many services in the Genesis II grid infrastructure support the ByteIO interface as a means of querying and manipulating various management aspects of its diverse set of services.

[6] http://ws.apache.org/axis/

d. FLE (clean room prototype)

Fujitsu Laboratories of Europe, Ltd. (FLE) participated in the OGSA-ByteIO Interoperability Testing with a clean-room prototype. This prototype was used as a real-life showcase to demonstrate that Fujitsu's implementation of a distributed service environment is usable, scalable and suitable for further development and progress of Fujitsu's implementation itself.

The prototype uses JAXB (Java API for XML Binding) [FIALLI, KAWAGUCHI] for marshalling and un-marshalling XML documents. The decision for JAXB is closely linked to the decision to use the Sun JAX-WS Reference implementation version 2.1.1 for the JAX-WS specification version 2.0; JAX-WS explicitly ad exclusively uses JAXB as its XML document handling framework.

Fujitsu's implementation makes use of the WS-Addressing final version (namespace "http://www.w3.org/2005/08/addressing") and uses reference parameters to disambiguate WS endpoints that implement the ByteIO Random Access port type. Fujitsu did not implement any other WS-Addressing profiles at this point, but is planning to extend its usage of WS-Addressing to complement with the WS-Naming specification.

Fujitsu focused on implementing the Random Access port type only in the prototype. Later versions of Fujitsu's distributed service environment will include the full implementation of the OGSA-ByteIO family of port types. Within the Random Access port type implementation, Fujitsu implemented all mandatory and optional elements of the specification, except for the "Last Access Time" Resource Property. The reason is that unless a native interface is provided, Java 1.5 does not support the POSIX file attribute of the last access time of a file.

3. Interoperability testing setup

The ByteIO Interoperability Fiesta was arranged as a "virtual meeting", rather than a physical face to face meeting. This allowed the actual time period of the fiesta to be spread over a longer period, and also made it possible for more groups to directly participate at a lower cost.

The fiesta was based on draft 10 of the ByteIO Interoperability Testing Specification document [DRESCHER]. During the course of the Interoperability Fiesta, the draft document was updated to incorporate corrections and clarifications raised by the interop process.

a. Process and Infrastructure

The ByteIO Interoperability Fiesta was announced to the OGSA® ByteIO Working Group mailing list on May 7, 2007 by the group chairs. Prospective participants were asked to indicate their willingness to participate, and to provide publicly accessible endpoints for their implementations by close of business on July 20, 2007. A wiki[7] was setup on GridForge to collect information relating to the Interoperability Fiesta and was updated through the period by all participants. Each participant indicated on the wiki when their endpoints were ready for testing.

The ByteIO mailing list was used for discussion of implementation and interoperability fiesta issues, and telephone conferences were scheduled where appropriate.

4. Interoperability testing results

a. General Status of the Services

	4.1 RByteIO GetResource Property	4.2, 4.3, 4.4 RByteIO	4.5, 4.6, 4.7 RByteIO	4.8 RByteIO Append	4.9 RByteIO Trunc	5.1 SByteIO GetResource Property	5.2 SByteIO SeekRead	5.3 SByteIO SeekWrite

7 http://forge.ogf.org/sf/wiki/do/viewPage/projects.byteio-wg/wiki/HomePage?showDetails=true

		Read	Write		Append			
UVa	ready	ready	ready	ready	ready	ready	ready	ready
Fujitsu	ready	ready	ready	ready	ready	not ready	not ready	not ready
EPCC	ready	ready	ready	ready	ready	not ready	not ready	not ready
FZJ	ready	ready	ready	ready	ready	ready	ready	ready

b. Interoperability Tables

These tables represent the results of the tests at the end of the Interoperability Fiesta. For each table, the test is given both a name, and a section number. The section numbers refer to the ByteIO Interoperability Testing Scenarios document [DRESCHER]. For each table, the following symbols may be entered for a cell:

- not tested – the current implementations of the client/service pair have not been tested
- success – the current implementations of the client/service pair have been tested successfully
- fail – the current implementations of the client/service pair have been tested and has failed

Clients are listed in the first column, services in the first row.

Section 4.1 [`RByteIO GetResourceProperty`]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	success	Success
Fujitsu	success	-	not tested	Success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.2 [`RByteIO Read i`]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	success	Success
Fujitsu	success	-	not tested	Success
EPCC	not tested	Not tested	-	not tested
FZJ	success	success	fail	-

Section 4.3 [`RByteIO Read ii`]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	success	success
Fujitsu	success	-	not tested	success
EPCC	not tested	Not tested	-	not tested
FZJ	success	success	fail	-

Section 4.4 [`RByteIO Read iii`]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	success	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.5 [`RByteIO Write i`]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**

UVa	-	success	fail	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.6 [RByteIO Write ii]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	fail	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.7 [RByteIO Write iii]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	fail	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.8 [RByteIO Append]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	fail	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 4.9 [RByteIO TruncAppend]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	success	fail	success
Fujitsu	success	-	not tested	success
EPCC	not tested	not tested	-	not tested
FZJ	success	success	fail	-

Section 5.1 [SByteIO GetResourceProperty]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	not tested	not tested	success
Fujitsu	not tested	-	not tested	not tested
EPCC	not tested	not tested	-	not tested
FZJ	success	not tested	not tested	-

Section 5.2 [SByteIO SeekRead]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**
UVa	-	not tested	not tested	success
Fujitsu	not tested	-	not tested	not tested
EPCC	not tested	not tested	-	not tested
FZJ	success	not tested	not tested	-

Section 5.3 [SByteIO SeekWrite]

Clients\Services	**UVa**	**Fujitsu**	**EPCC**	**FZJ**

UVa	-	not tested	not tested	success
Fujitsu	not tested	-	not tested	not tested
EPCC	not tested	not tested	-	not tested
FZJ	success	not tested	not tested	-

c. Issues discovered in ByteIO Functional Specification 1.0

None noted.

d. Issues discovered in ByteIO WSRF Rendering Specification 1.0

Missing ResourceUnavailableFaultByteIO RandomByteIO portType

Issue: The mandatory fault "wsrf-rpw:ResourceUnavailableFault" has not been specified on the portType operations "wsrf-rp:GetResourceProperty", "wsrf-rp:GetMultipleResourceProperties" and "wsrf-rp:QueryResourceProperties".

Resolution: Add `<wsdl:fault name="ResourceUnavailableFault" message="wsrf-rw:ResourceUnavailableFault"/>` to each operation.

e. Issues discovered in ByteIO Interoperability Testing Specification (draft 10)

Missing Tags in SOAP Body of Request Message
(4.2) wsrf-rp:GetMultipleResourceProperties operation

Issue: The non-normative example of the elements the ByteIO client adds to the SOAP body of the request message is missing the <wsrf-rp:ResourceProperty> tag.

Resolution: The SOAP body should instead contain the following:

```
<wsrf-rp:GetMultipleResourceProperties>
   <wsrf-rp:ResourceProperty> rbyteio:Readable </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty> rbyteio:Writeable </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty> rbyteio:TransferMechanism </wsrf-rp:ResourceProperty>
</wsrf-rp:GetMultipleResourceProperties>
```

Incorrect Resource Property Name
(4.3) wsrf-rp:QueryResourceProperties operation

An incorrect name for the Resource Property is used in the non-normative example of the elements the ByteIO client adds to the SOAP body of the request message. The Resource Property queried is called “ModificationTime” not “ModificationDate”.

Resolution: The SOAP body should instead contain the following:

```
<wsrf-rp:QueryResourceProperties>
   <wsrf-rp:QueryExpression Dialect=”http://www.w3.org/TR/1999/REC-xpath-19991116”>
      /*/rbyteio:ModificationTime
   </wsrf-rp:QueryExpression>
</wsrf-rp:QueryResourceProperties>
```

Ambiguity: Queries and Namespaces

(4.3) wsrf-rp:QueryResourceProperties operation

Issue: If the client wishes to query a resources property, should the query contain the namespace? Which of the following queries are valid: “/*/rbyteio:ModificationTime” vs. “/*/ModificationTime”?

Missing Tag in SOAP Body of Response Message

(4.3) wsrf-rp:QueryResourceProperties operation

Issue: The non-normative example of the elements the ByteIO client adds to the SOAP body of the response message is missing the <rbyteio:ModificationTime> tag.

Resolution: The SOAP body should instead contain the following:

```
<wsrf-rp:QueryResourcePropertiesResponse>
   <rbyteio:~ModificationTime>
     2006-09-11T16:15:33+05:00
   </rbyteio:ModificationTime>
</wsrf-rp:QueryResourcePropertiesResponse>
```

Missing Tags in SOAP Body of Request Message

(5.2) wsrf-rp:GetMultipleResourceProperties operation

Issue: The non-normative example of the elements the ByteIO client adds to the SOAP body of the request message is missing the <wsrf-rp:ResourceProperty> tag.

Resolution: The SOAP body should instead contain the following:

```
<wsrf-rp:GetMultipleResourceProperties>
   <wsrf-rp:ResourceProperty> sbyteio:Seekable </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty> sbyteio:TransferMechanism </wsrf-rp:ResourceProperty>
   <wsrf-rp:ResourceProperty> sbyteio:EndOfStream </wsrf-rp:ResourceProperty>
</wsrf-rp:GetMultipleResourceProperties>
```

Ambiguity: Queries and Namespaces

(5.3) wsrf-rp:QueryResourceProperties operation

Issue: This section incorrectly refers to a modification date query. It should instead state: In this case, the client wishes to query the resource’s writeable property. Mainly, it is unclear whether the non-normative example of the elements the ByteIO client adds to the SOAP body of the response message contains the correct query.

Resolution: If the query should include the namespace, then it should be “/*/sbyteio:Writeable” and the SOAP body should then contain the following:

```
<wsrf-rp:QueryResourceProperties>
   <wsrf-rp:QueryExpression Dialect=”http://www.w3.org/TR/1999/REC-xpath-19991116”>
      /*/sbyteio:Writeable
   </wsrf-rp:QueryExpression>
</wsrf-rp:QueryResourceProperties>
```

If the namespace is not required, then querying for “/*/Writeable” is fine.

Missing Tag in SOAP body of Response Message
(5.3) wsrf-rp:QueryResourceProperties operation

Issue: The non-normative example of the elements the ByteIO client adds to the SOAP body of the response message is missing the <sbyteio:Writeable> tag.

Resolution: The SOAP body should instead contain the following:

```
<wsrf-rp:QueryResourcePropertiesResponse>
   <sbyteio:Writeable>
     true
   </sbyteio:Writeable>
</wsrf-rp:QueryResourcePropertiesResponse>
```

WS-I compliance violations and missing SOAP bindings
Appendix B: Normative RandomByteIO Binding for Interoperability Tests

There are a number of WS-I compliance violations and missing SOAP binding information from the normative transport binding to be used in the Interoperability Tests provided in Appendix B.

Issue: The input soap:body child element's attribute "namespace" violates WS-I compliance rules. Operations affected: "wsrf-rpw:GetResourceProperty", "wsrf-rpw:GetMultipleResourceProperties", "wsrf-rpw:QueryResourceProperties", "rbyteio:write", "rbyteio:append", "rbyteio:truncAppend".

Resolution: Remove the namespace attribute from the "soap:body" element

Issue: The output soap:body child element's attribute "namespace" violates WS-I compliance rules.
Operations affected: "wsrf-rpw:GetResourceProperty", "wsrf-rpw:GetMultipleResourceProperties", "wsrf-rpw:QueryResourceProperties", "rbyteio:write", "rbyteio:append", "rbyteio:truncAppend".

Resolution: Remove the namespace attribute from the "soap:body" element

Issue: The binding does not specify a SOAP binding for the "wsrf-rpw:ResourceUnavailableFault". Operations affected: "wsrf-rpw:GetResourceProperty", "wsrf-rpw:GetMultipleResourceProperties", "wsrf-rpw:QueryResourceProperties".

Resolution: Add appropriate binding information

f. Issues discovered in ByteIO Interoperability Testing Specification (draft 11)

Incorrect Write Value of *Transfer-Information*
(4.5) rbyteio:write operation (i)
(5.3.1) sbyteio:seekWrite operation

Issue: "KysrKysr" is the value that corresponds to writing a block of bytes equivalent to "++++++". This value is incorrectly given as "KysrKys=" in Section 4.5 and 5.3.1.

Resolution: The corrected SOAP request messages follow:

- 4.5 The client MUST use "KysrKysr" as value for rbyteio:write/rbyteio:transfer-information

```
<rbyteio:write>
   <rbyteio:start-offset> 20 </rbyteio:start-offset>
   <rbyteio:bytes-per-block> 6 </rbyteio:bytes-per-block>
   <rbyteio:stride> 0 </rbyteio:stride>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> KysrKysr </byteio:data>
   </rbyteio:transfer-information>
</rbyteio:write>
```

- 5.3.1 The client MUST use "KysrKysr" as value for sbyteio:seekWrite/sbyteio:transfer-information

```
<sbyteio:seekWrite>
   <sbyteio:offset> 20 </sbyteio:offset>
   <sbyteio:seek-origin>
      http://schemas.ggf.org/byteio/2005/10/streamable-access/seek-origins/beginning
   </sbyteio:seek-origin>
   <sbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> KysrKys= </byteio:data>
   </sbyteio:transfer-information>
</sbyteio:seekWrite>
```

Wrong Treatment of Transfer-Information-Type XML element
Occurs throughout document

The Interoperability Testing Specification repeatedly and consistently uses wrong "transfer-information-type" XML elements in both sample XML fragments, and in its interoperability requirements.

For example, the document gives as sample SOAP fragment:

```
<rbyteio:readResponse>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      MTExMjEzMTQxNTE2
   </rbyteio:transfer-information>
</rbyteio:readResponse>
```

but it MUST read:

```
<rbyteio:readResponse>
   <rbyteio:transfer-information transfer-
mechanism="http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data>MTExMjEzMTQxNTE2</byteio:data>
   </rbyteio:transfer-information>
</rbyteio:readResponse>
```

This mistake is repeated in many sections of the document.

Resolution: Per Appendix C of the ByteIO Recomendation Document, a <byteio:data> tag should surround data being passed. This applies to response messages of read operations and request messages of write operations. (Request messages of read operations and response messages of write operations should not contain data.)

The corrected SOAP body messages of the affected tests are listed below:

- 4.2 Read Operation Response (i)

```
<rbyteio:readResponse>
   <rbyteio:transfer-information transfer-
mechanism="http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> MTExMjEzMTQxNTE2 </byteio:data>
   </rbyteio:transfer-information>
</rbyteio:readResponse>
```

- 4.3 Read Operation Response (ii)

```
<rbyteio:readResponse>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> MDIwNDA2MDgxMA==  </byteio:data>
   </rbyteio:transfer-information>
</rbyteio:readResponse>
```

- 4.4 Read Operation Response (ii)

```
<rbyteio:readResponse>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> MDEwMjAzMDQwNTA2MDQwNTA2MDcwODA5 </byteio:data>
   </rbyteio:transfer-information>
</rbyteio:readResponse>
```

- 4.5.1 Write Operation Request (i)

```
<rbyteio:write>
   <rbyteio:start-offset> 20 </rbyteio:start-offset>
   <rbyteio:bytes-per-block> 6 </rbyteio:bytes-per-block>
   <rbyteio:stride> 0 </rbyteio:stride>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
      <byteio:data> KysrKysr </byteio:data>
   </rbyteio:transfer-information>
</rbyteio:write>
```

- 4.6.1 Write Operation Request (ii)

```
<rbyteio:write>
   <rbyteio:start-offset> 22 </rbyteio:start-offset>
   <rbyteio:bytes-per-block> 2 </rbyteio:bytes-per-block>
   <rbyteio:stride> 4 </rbyteio:stride>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
       <byteio:data>  KysrKysrKysrKw== </byteio:data>
    </rbyteio:transfer-information>
</rbyteio:write>
```

- 4.7.1 Write Operation Request (iii)

```
<rbyteio:write>
    <rbyteio:start-offset> 0 </rbyteio:start-offset>
    <rbyteio:bytes-per-block> 6 </rbyteio:bytes-per-block>
    <rbyteio:stride> 3 </rbyteio:stride>
    <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
        <byteio:data> Pz8/Pz8/KysrKysr </byteio:data>
    </rbyteio:transfer-information>
</rbyteio:write>
```

- 4.8.1 Append Operation Request

```
<rbyteio:append>
    <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
        <byteio:data> KysrKysr </byteio:data>
    </rbyteio:transfer-information>
</rbyteio:append>
```

- 4.9.1 TruncAppend Operation Request

```
<rbyteio:truncAppend>
    <rbyteio:offset> 30 </rbyteio:offset>
    <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
        <byteio:data> KysrKysr </byteio:data>
    </rbyteio:transfer-information>
</rbyteio:truncAppend>
```

- 5.2 SeekRead Operation Response

```
<sbyteio:seekReadResponse>
    <sbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple">
        <byteio:data> MTExMjEzMTQxNTE2 </byteio:data>
    </sbyteio:transfer-information>
</sbyteio:seekReadResponse>
```

ByteIO Interop Port Type, DeleteResource, and Race Conditions

Appendix A: Normative Interoperability Interface

Issue: Per Appendix A of the ByteIO Interoperability Testing Scenarios, the ByteIOInterop::PortType defines the *DeleteResource* operation as a one-way message. This results in a race condition if a resource is deleted and then tested for successful deletion.

g. Issues discovered in ByteIO Interoperability Testing Specification (draft 12)

Incorrect Blocks-per-Byte in SOAP Example
(4.2) rbyteio:read operation (i)

Issue: The non-normative example of the SOAP elements in the body of the request message does not match the specifications of test 4.2. The required bytes-per-block is 12 not 6.

Resolution: The SOAP body of the request message should be as follows:

```
<rbyteio:read>
   <rbyteio:start-offset> 20 </rbyteio:start-offset>
   <rbyteio:bytes-per-block> 6 </rbyteio:bytes-per-block>
   <rbyteio:num-blocks> 1 </rbyteio:num-blocks>
   <rbyteio:stride> 0 </rbyteio:stride>
   <rbyteio:transfer-information transfer-mechanism=
"http://schemas.ggf.org/byteio/2005/10/transfer-mechanisms/simple"/>
</rbyteio:read>
```

h. Issues discovered in ByteIO Interoperability Testing Specification (draft 13)

Update to Interop Factory WSDL
Appendix A: Normative Interoperability Interface

Issue: The original definition of the "createResource" operation on the interop factory port type caused problems with various WS tool that are in use. Particularly, some tools had problems supporting the "xsi:nillable" attribute. As a consequence, testing was not unreliable as sometimes testable resources were created, and sometimes not.

```
         <xsd:element name="createResource" nillable="true"/>
```

Resolution: the Interop Factory WSDL was updated to contain the following createResource element:

```
         <xsd:element name="createResource">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name=”dummy” type=”xsd:int” minOccurs=”1” maxOccurs=”1”/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
```

i. Issues discovered in ByteIO Interoperability Testing Specification (draft 14)

None reported

5. Discussion

This section contains a discussion of the results of the Interoperability in the form of individual comments from each of the participants. The remarks in the individual participants sections have only been endorsed by that participant. The remarks made in the conclusions which follows have been endorsed by all participants.

a. FZJ specific discussion

FZJ used an implementation of the ByteIO specification to expose one of the UNICORE file transfer interfaces. Although it covers only small portion of UNICORE functionality, the scope and possibility for usage in conjunction with other UNICORE core services are significant. Specifically, ByteIO implementation currently leverages UNICORE's storage and job management services.

The ByteIO interoperability fiesta led to some improvements with respect to the standards compliance and implementation; moreover the defined test cases were more pragmatic and helpful while identifying major and/or minor problems. With the FZJ implementation, the majority of the problems identified were pertaining to the proper use of XML messaging and WS-Addressing types rather than issues with compliance to the specification. Although the implementation of the ByteIO specification was relatively trivial in terms of development effort and time, the interoperability exercise helped in refining the hosting environment implementation.

b. EPCC specific discussion

The main problems encountered by EPCC during the implementation phase were caused by the tooling. For example, OGSA-DAI uses a different version of WS-Addressing. Unfortunately the EPCC implementation could not be fully tested as there were problems with the tooling when accessing the interop factory. In the OGSA-DAI implementation a ByteIO resource is created by submitting an activity for execution on the OGSA-DAI execution resource. This activity creates a resource. Data can be written to and read from the resource via the RandomByteIO interface or by using OGSA-DAI activities. For interoperability testing this approach was not suitable because clients that do not support OGSA-DAI requests (i.e. all other clients involved in the interop experiment) cannot create ByteIO resources. Therefore a factory had to be provided specifically for creating ByteIO resources. The interop testing failed mainly because the interop factory interface was not accessible by some of the tooling. A modification of the factory WSDL was suggested but not implemented by EPCC until after the conclusion of the interop tests.

c. UVa specific discussion

The ByteIO specification was very clear and easy to implement (taking less than a day for each port type), with the main barrier to interoperation being WS-Addressing. Specifically, WS-Addressing has been, and continues to be a substantial barrier to interoperability success (followed closely by WSDL and XSD themselves). The core problem is that as a specification, WS-Addressing tends towards the complex and as a result various implementations do not always fully implement the specification correctly. Further, while the specification specifically allows for a web service endpoint to be referenced by an EndpointReferenceType (EPR) that utilizes both the Address field and the ReferenceParameters field, specific grid implementations have in the past had a tendency to assume only the former of these during addressing. Problems with WSDL and XSD boil down to the same, i.e., that as parseable languages, they tend towards complexity, thus increasing the chances of any given implementation to contain bugs and deviations from the specifications (reminiscent of the problems encountered later in the life of C++ as the language became unnecessarily complex and thus leading to the near certainty that no two C++ compilers would ever recognized the exact same language). Another sticky point for the UVa implementation was WSRF-RP, but this turned out not to be a big problem. Overall experience with this interoperability exercise was favorable and it was not nearly as hard as some interoperability exercises for other specifications have been.

The RandomByteIO interface has proven invaluable in the Genesis II system. Along with RNS, it is used through the system on every service implemented for management and user interface. The StreamableByteIO interface is also extremely valuable, but not for day-to-day uses. It turns out to be useful when one wants to quickly open a ByteIO stream to a service for management purposes (for example, using StreamableByteIO to stream JSDL documents down to the Genesis II BES implementation thus providing the ability to start jobs using UNIX cp, Windows drag-and-drop). In order to make StreamableByteIO even more valuable to users as a whole, it would be nice to have a common port type for "opening" a stream.

d. FLE specific discussion

The ByteIO specification is a straightforward specification that is easily implemented within one week at most given basic experience in WS based service programming and XML processing. If integrating with existing implementations, however, it may require more time to properly implement the ByteIO port types as part of the overall system, but those alterations are expected to be mostly focused on code changes not related to ByteIO itself.

The interoperability testing itself revealed only minor issues. The important point to note is that those issues were not ByteIO related. Instead those issues included general XML and WS message processing and are thus targeted at the toolkits that are used to implement the ByteIO port types rather than ByteIO itself.

6. Conclusion

The ByteIO interoperability fiesta success shows that the specification describes a pair of port types which, with minor fixes as indicated in this document, can be implemented by separate organizations in an unambigous (with respect to interface and port type) way. These grids can, together with other specifications provided by the OGF (for example, the OGSA-WSRF-BP), can then be used interoperably by users. Further, the interoperability fiesta also shows that virtual interoperability festivals are feasible and under the right circumstances can be used effectively.

It's worth noting that the task taken on by Michel Drescher to provide an interoperability test document which was then vetted by the standard OGF document process proved invaluable in making the ByteIO Interoperability Fiesta a success. His careful attention to detail allowed for the fiesta participants to provide rigorous tests that could easily be validated and "graded" for success.

If there were any negative aspects to the interoperability fiesta, they would be along the lines of the standard problems that seem inherent in OGF and web services interop festivals in general. Namely, the specifications themselves rely on tooling, core specifications, and other 3rd party products that tend to hamper success. WS-Addressing and WSDL are both complex specifications that have a tendency to promote only partially correct implementations. Fiesta participants often rely on tooling to manipulate these specifications and end doing so can end up becoming tied to a fundamentally flawed tool or library. It has been our experience that no grid implementation is free from this particular problem and the fact that the issues seem to continually raise their ugly heads indicates a fundamental flaw in either the process, or the foundations on which the grid services world has built its specifications.

In summary, this particular interop fiesta has shown that ByteIO is a reasonable and implementable specification that promised good interoperability. Many of the projects included have shown that the specification itself is also useful to grid implementers and presumably to their target users.

7. Security Considerations

The Interoperability Fiesta was carried out without any security mechanisms, mainly to ensure that the fiesta can focus on interoperability problems on the BYTEIO family of port types. This was possible as the ByteIO Functional Specification 1.0 as well as the WSRF Rendering Specification do not provide any factory like pattern to create or obtain EPRs to resources that implement the ByteIO port types. In real life, those factory patterns are mostly implementation specific or deployment specific, allowing the interop participants to agree on an interop specific factory. An interop specific factory in turn allows to apply very restrictive resource consumption policies once the backing resources were created.

For example, the FLE implementation, instead of backing the ByteIO resource with a real file on the file system or an entry in an RDBMS, used a special in-memory backing implementation with restricted memory limits when the ByteIO resource was created. Additionally, the resources were given very short lifetimes of 5 minutes maximum. However, while the backing mechanism was special, the WS message-processing layer was the same altogether.

The OGSA-ByteIO WG strongly recommends to not use the ByteIO port types as-is to deploy Grid resources. The port types are designed so that they can be composed with other port types, message-level and transport-level mechanisms. For example, combining the ByteIO port types with transport level security (HTTPS, etc.) or message level security (e.g. WS-Security and associated token profiles).Having said that, the existing OGSA Security Profiles as well as the currently developed Security Profiles should be considered when implementing and deploying OGSA-ByteIO.

8. Contributors

Neil Chue Hong (editor)
OMII-UK
59/3235
University of Southampton
Southampton SO17 1BJ
United Kingdom

Michel Drescher
Fujitsu Laboratories of Europe Limited
Hayes Park Central
Hayes End Road
Hayes
Middlesex UB4 8FE
United Kingdom

Amy Krause
EPCC
The University of Edinburgh
JCMB, Mayfield Road
Edinburgh EH9 3JZ
United Kingdom

M. Shahbaz Memon
Department of Applied Mathematics
Forschungszentrum Jülich GmbH
Wilhelm-Johnen-Str. 1

D–52425 Jülich
Germany

Mark Morgan
Department of Computer Science
University of Virginia
151 Engineer’s Way
P.O. Box 400740
Charlottesville
VA. 22904-4740
United States of America

We gratefully recognize the contributions of Odysseas Bournas (University of Edinburgh) and Karolina Sarnowska (University of Virginia) in the development of implementations of ByteIO. We also recognize the contributions of Dave Snelling (Fujitsu Laboratories of Europe), Andrew Grimshaw (University of Virginia) and A. Shiraz Memon (Forschungszentrum Jülich) in commenting on this work.

The work at EPCC, The University of Edinburgh was supported by EPSRC through OMII-UK and through the UoE MSc in HPC programme.

9. Intellectual Property Statement

10. Disclaimer

11. Full Copyright Notice

12. References

Note that only permanent documents should be cited as references. Other items, such as Web pages or working groups, should be cited inline (i.e., see the Open Grid Forum, http://www.ogf.org). References should conform to a standard such as used by IEEE/ACM, MLA, Chicago or similar. Include an author, year, title, publisher, place of publication. For online materials, also add a URL. It is acceptable to separate out "normative references," as IETF documents typically do. Some sample citations:

[BOX] Box, D. et al. Web Services Addressing (WS-Addressing). W3C Member Submission. August 2004.
[BRADNER1] Bradner, S. Key Words for Use in RFCs to Indicate Requirement Levels, RFC 2119. March 1997.
[BRADNER2] Bradner, S. The Internet Standards Process – Revision 3, RFC 2026. October 1996.
[CATLETT] Catlett, C. GFD-1: Grid Forum Documents and Recommendations: Process and Requirements. Lemont, Illinois: Global Grid Forum. April 2002.
[DRESCHER] Drescher, M. OGSA®-ByteIO Interoperability Testing Specification (Draft 10). Open Grid Forum. October 2006.
[FIALLI] Fialli, J. et al. JSR 31: XML Data Binding Specification. Java Community Process. March 2003.
[GUDGIN1] Gudgin, M. et al. Web Services Addressing 1.0 – Core. W3C Recommendation. May 2006.
[GUDGIN2] Gudgin, M. et al. Web Services Addressing 1.0 – SOAP Binding. W3C Recommendation. May 2006.
[GUDGIN3] Gudgin, M. et al. Web Services Addressing 1.0 – Metadata. W3C Recommendation. September 2007.
[KAWAGUCHI] Kawaguchi, K. et al. JSR 222: Java™ Architecture for XML Binding (JAXB) 2.0. Java Community Process. December 2006.
[MORGAN1] Morgan, M. GFD.87 ByteIO Specification 1.0. Open Grid Forum. January 2007.
[MORGAN2] Morgan, M. GFD.88 ByteIO OGSA® WSRF Basic Profile Rendering 1.0. Open Grid Forum. January 2007.
[RESCORLA] Rescorla, E. Guidelines for Writing RFC Text on Security Considerations. RFC 3552. July 2003.