# Describing ServiceData in Web Services

**(Authors Alphabetically)**
{Steve Graham, Martin Nally, Frank Leymann, Ellen Stokes, Tony Storey, Sanjiva Weerawarana}[1] { Karl Czajkowski}[2] {Steve Tuecke}[3]
[1] IBM
[2] USC/ISI
[3] Argonne National Labs

12~~1~~/~~26~~01/2002

## 1 ServiceData

The approach to *stateful* Web services introduced in the Grid services specification [http://www.ggf.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-05_2002-11-04.pdf] identified the need for a common mechanism to expose a service instance's state data to service requestors for query and change notification. The term used is "serviceData".

Since this concept is applicable to any Web service including those used outside the context of Grid applications, we propose a common approach to exposing Web service state data called serviceData.

The structure of this document is as follows: in the next section we discuss the motivation for serviceData, following that, we give details on how to describe serviceData within a portType, including a brief example of its use from the Grid services specification, and a discussion additional aspects of serviceData.

~Insert Frey's treatise on serviceData, and upon said insertion, include Frey's name on author list.

## 2 Motivation

The serviceData concept was introduced to provide a flexible, properties-style approach to access state data of a Web service. We developed this approach based on scenarios from systems management and the grid. These domains exhibit entities that have a potentially large number of properties. If we wish to model these entities using Web services, we need an approach that models this complex state effectively.

An alternative approach, of providing "get operation per property" was considered, but not adopted for several reasons:

1. If we use a "get" operation per property, then the set of messages to get multiple properties is chatty, does not allow atomicity to be addressed, and does not allow queries across multiple properties. Example: A Web service modeling an operatingSystem object contains (among other properties) the following four properties: totalVirtualMemorySize, freeVirtualMemory, freePhysicalMemory,

totalVisibleMemorySize. Two of the properties, totalVirtualMemorySize and totalVisibleMemory, are constant and set at boot time. The other two properties, freeVirtualMemory and freePhysicalMemory, change over time as memory is allocated and de-allocated to a process. Consider the case wherein a management application is monitoring (and displaying) memory usage, totalVirtualMemorySize and totalVisibleMemorySize can be read individually since both are constant in a running system. Since freeVirtualMemory and freePhysicalMemory do change and change with respect to each other, these two properties need to be obtained together atomically in a single get operation; otherwise, the usage displayed of physical and virtual memory for a given point in time is incorrect. We could just define an operation that does a get on both freeVirtualMemory and freePhysicalMemory, but this approach does not scale with the number of properties (see point 2.).

2. If we then compel portType designers to define additional operations to get all the combinations of the serviceData, the number of operations is voluminous. Although the above example is simple, in reality there are many other properties in the operatingSystem object, many of which have exactly the same behavior described in the example in 1. and depending on exactly what you monitor from the complete operatingSystem object would require all combinations of dynamically changing properties, e.g. 10 properties (from OperatingSystem) would yield 10! operations.

By supporting a serviceData approach, we provide the mechanism for requestors(e.g. example management applications) to build query expressions (eg XPath and XQuery) to express queries involving combinations of service data elements. Example: The operatingSystem resource contains (among other properties) the following three properties: totalSwapSpaceSize, sizeStoredInPagingFiles, and freeSpaceInPagingFiles. The filesystem resource contains (among other properties) the following two properties: fileSystemSize and availableSpace. These two resources are related since filesystem is contained in or booted from operating system. The properties fileSystemSize and totalSwapSpaceSize are typically constant, even between system boots. The other properties in this example change over time and with respect to each other, e.g. as sizeStoredInPagingFiles increases, freeSpaceInPagingFiles and availableSpace decreases. To monitor filesystem usage and display that usage, the query spans instances of both objects (operatingSystem and filesystem) and multiple properties within those instances. Single 'get' operations does not solve the problem of retrieving data across multiple object instances atomically.

There is also a requirement to support dynamic addition of "properties" at runtime. The notion is that the interface (portType) defines the majority of the serviceData elements (properties), however it is possible that at runtime, perhaps associated with a particular lifecycle state, to add serviceData elements to the Web service instance. Example: A Web service contains its lifecycle state in a serviceData element where the possible lifecycle values are 'exists', 'running', 'maintenance', and 'failed'. There is additional and different serviceData associated with each of these lifecycle states. In the failed lifecycle state, there is an additional serviceData element that contains the 'debug stack'. In the maintenance lifecycle state, the additional serviceData element is the list of data to help process the maintenance state. The 'exists' and 'running' lifecycle states have no additional serviceData elements. In Web services, when serviceData elements exist in a portType, they are expected to be valid. In this example where the information associated with a lifecycle state is variable, use of dynamic serviceData for access to that additional information is a good solution because the right needed information is returned based on lifecycle state. A counter argument would be to define an operation 'getLifecycleState' to accomplish the same, but it is not the state itself that that operation would get, but rather the five (or however many) new serviceData elements associated with that new state; you wouldn't want to model the aggregation of those five properties into a getLifecycleState given the variability of serviceData information associated with each lifecycle state.

## 3 Extending PortType with WS-ServiceData

ServiceData defines a new child element of a portType named **serviceData**. This element defines serviceData elements associated with that portType. Optionally, initial values for those serviceData elements can be specified.

```
   <wsdl:portType name="ncname"> *
      <wsdl:documentation .... /> ?
      <wsdl:operation name="ncname"> *
…
      <wsdl:serviceData name="ncname" … /> *
      <wsdl:staticServiceDataValues>?
         <some element>*
      </wsdl:staticServiceDataValues>
…
   </wsdl:portType>
```

For example, a portType declares a set of serviceData elements such as shown below:

```
<wsdl:definitions xmlns:tns=”xxx” targetNamespace=”xxx”>
   <wsdl:portType name="exampleSDUse"> *
      <wsdl:operation name=…>
…
      <wsdl:serviceData name="sd1" type=”xsd:String”
                        mutability=”static”/>
      <wsdl:serviceData name="sd2" type=”tns:SomeComplexType”/>
…
      <wsdl:staticServiceDataValues>
         <xxx:sdl>initValue</xxx:sd1>
      </wsdl:staticServiceDataValues>
   </wsdl:portType>
…
</wsdl:definitions>
```

Initial values for “static” serviceData elements can also be declared in the portType.

Any service that implements the portType named “exampleSDUse” MUST have as part of its state, serviceData elements with qualified names “tns:sd1” and “tns:sd2”. This concept is similar to the notion of an instance variable in Object-Oriented programming languages such as Java $^{tm}$, Smalltalk and C++ and most closely associated with a Java Bean property.

## 3.1 Structure of the ServiceData Element Declaration

The definition for serviceData element is:

```
<xs:complexType name=“ClosedServiceDataElementType”>
   <xs:complexContent>
      <xs:restriction base=”xs:element”>
         <xs:attribute name="name" type="xs:NCName"/>
         ~~<xs:attributeGroup ref="xs:defRef" /> (name and ref)~~
         <xs:attribute name="type" type="xs:QName" />
         <xs:attributeGroup ref="xs:occurs" /> (minOccurs, maxOccurs)
         <xs:attribute name="nillable" type="xs:boolean" use="optional"
                       default="false" />
         <xs:anyAttribute namespace="##other" processContents="lax" />
      </xs:restiction>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name=”ServiceDataType”>
   <xs:complexContent>
      <xs:extension base=”ClosedServiceDataElementType”>
         <xs:sequence>
            <xs:any namespace="##other" minOccurs="0"
                                        maxOccurs="unbounded" />
         </xs:sequence>
         <xs:attribute name="mutability" default=”extend”>
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="static"/>
                  <xs:enumeration value="constant"/>
                  <xs:enumeration value="extend"/>
                  <xs:enumeration value="mutable"/>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="modifiable" type=”xs:boolean”
                                         default=”false”>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>

<element name=”serviceData” type=”ServiceDataType”/>

<xs:complexType name=”ServiceDataValuesType”>
   <xs:complexContent>
      <xs:sequence>
         <xs:any namespace="##any" minOccurs="0"
                                   maxOccurs="unbounded" />
      </xs:sequence>
   <xs:complexContent>
</xs:complexType>
```

```
<element name=”serviceDataValues” type=”ServiceDataValuesType”/>
```

In the above approach, a serviceData element is restricted to contain only these properties from xsd:element:

- name
- type
- minOccurs
- maxOccurs
- nillable
- open attribute content model

And the model is extended to allow open element content model, and to define the mutability attribute (3.7) and the modifiable attribute.

We describe how the concept of serviceData can be exposed through operations in section 3.5. In a related document “Grid Services Operations on ServiceData”, we describe one convention on how the serviceData element values can be queried and set by service requestors and how service requestors can be notified of a change in the serviceData element’s value.

## 3.2 Using ServiceData, an Example from Grid Services,

Lets examine how ServiceData can be used by reviewing an example currently documented in the Grid Services Specification. The Grid Service Specification defines a portType, named “GridService” (in a namespace that is commonly associated with the prefix *gsdl*) that acts as the base for all other Grid service portTypes (similar to “Object” class in Java or Smalltalk). The serviceData elements declared for the Grid Service portType are shown below:

```
<wsdl:definitions …
   xmlns:gsdl=
      "http://www.gridforum.org/namespaces/2002/11/gridServices"
   targetNamespace=
      "http://www.gridforum.org/namespaces/2002/11/gridServices"
   xnlns="http://www.gridforum.org/namespaces/2002/11/gridServices">

<wsdl:portType name=”GridService”>
   <wsdl:operation name= …>
   …

   <wsdl:serviceData name=”PortType” type=”gsdl:portTypeNameType”
        minOccurs=”1” maxOccurs=”unbounded”
        mutability=”constant”/>
   <wsdl:serviceData name=”serviceDataName” type=”xsd:QName”
        mutability=”mutable” maxOccurs=”unbounded”/>
   <wsdl:serviceData name=”FactoryHandle”
        type=”gsdl:gridServiceHandleType”
        minOccurs=”0” mutability=”constant”/>
   <wsdl:serviceData name=”GridServiceHandle”
        type=”gsdl:gridServiceHandleType”
        minOccurs=”0” maxOccurs=”unbounded”
        mutability=”extend”/>
   <wsdl:serviceData name=”GridServiceReferences”
        type=”gsdl:gridServiceReferenceType”
        minOccurs=”0” maxOccurs=”unbounded”
        mutability=”mutable”/>
   <wsdl:serviceData name=”QueryExpressionType”
        type=”gsdl:queryExpressionType”
        minOccurs=”1” maxOccurs=”unbounded”
        mutability=”extend”/>
   <wsdl:serviceData name=”TerminationTime” type=”gsdl:terminationTime”
        mutability=”mutable”/>
   <wsdl:serviceData name="CurrentTime" type="gsdl:currentTime"
        mutability="mutable"/>
…
```

And an example set of serviceData element values for some Web service instance might look like:

```
…
   xmlns:gsdl=
      "http://www.gridforum.org/namespaces/2002/11/gridServices"
   xmlns:crm=”http://gridforum.org/namespaces/2002/11/crm”
   xmlns:tns=”http://example.com/exampleNS”
   xnlns=”http://example.com/exampleNS”>

<wsdl:serviceDataValues>
   <gsdl:PortType>crm:GenericOSPT</gsdl:PortType>
   <gsdl:PortType>gsdl:GridService</gsdl:PortType>

   <gsdl:serviceDataName>gsdl:PortType
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:gsdlserviceDataName
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:FactoryHandle
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:gsdlGridServiceHandle
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:GridServiceReferences
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:QueryExpressionType
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:TerminationTime
   </gsdl:serviceDataName>
   <gsdl:serviceDataName>gsdl:CurrentTime
   </gsdl:serviceDataName>

   <gsdl:FactoryHandle>someURI</gsdl:FactoryHandle>

   <gsdl:GridServiceHandle>someURI</gsdl:GridServiceHandle>
   <gsdl:GridServiceHandle>someOtherURI</gsdl:GridServiceHandle>

   <gsdl:GridServiceReference>…</gsdl:GridServiceReference>
   <gsdl:GridServiceReference>…</gsdl:GridServiceReference>

   <gsdl:QueryExpressionType>
      gsdl:queryByServiceDataName
   </gsdl:QueryExpressionType>

   <gsdl:QueryExpressionType>
      gsdl:queryByMultipleServiceDataNames
   </gsdl:QueryExpressionType>

   <gsdl:TerminationTime after=”2002-11-01T11:22:33”
                         before=”2002-12-09T11:22:33”/>

   <gsdl:CurrentTime>2002-11-02T12:22:00</gsdl:CurrentTime>

</wsdl:serviceDataValues>
```

## 3.3 *Interpretation of the ServiceData Element Declaration*

The declaration of a serviceData element is very similar to the declaration of an element in XML Schema. Therefore we use a restriction of xsd:element syntax from XML Schema to declare serviceData elements.

- maxOccurs = (nonNegativeInteger | unbounded) : 1
  - This value indicates the maximum number of times the serviceData element can appear in the service instance’s serviceDataValues
- minOccurs = nonNegativeInteger : 1
  - This value indicates the minimum number of times the serviceData element can appear in the service instance’s serviceDataValues
  - If the value is 0, then the serviceData element is optional
- name = NCName and {target namespace}
  - The name of the serviceData element must be unique amongst all serviceData element declarations in the target namespace of the wsdl:definitions element.
  - The combination of the name of the serviceData element and the target namespace of the wsdl:definitions element forms a QName, allowing a unique reference to this serviceData element.
- nillable = boolean : false
  - 
  - Indicates whether the serviceData element can have a nil value (that is a value that has an attribute xsi:nil with value=”true”
    - For example a serviceData element

      ```
      <serviceDataElement name=”foo” type=”xs:string”
      nillable=true” />
      ```
    - can have a valid serviceDataElement value

      ```
      <foo xsi:nil=”true”/>
      ```
- type = QName
  - Defines the type of the serviceData element
- modifiable = “boolean” : false
  - If true, it is legal for requestors to update the serviceData value through operations, subject to constraints on cardinality (minOccurs, maxOccurs) and mutability. If false, the serviceData element should be regarded as “read only” by the requestor.

- mutability = “static” | “constant” | “extend” | “mutable” : extend
  - An indication on how the values of a serviceData element can change. See (section 3.7)
- {any attributes with non-schema namespace}
  - Open content on the attributes of serviceData declaration will be allowed.
- Content
  - Open content element model, meaning elements from any other namespace (besides XML Schema) may appear as child elements of the serviceData element.

## 3.4 ServiceData Values

Each service instance is associated with a collection of serviceData elements. The set of serviceData elements are defined within the various portTypes that form the interface of the service and may also be dynamically extended to include additional serviceData elements (see3.10~~3.9~~). We call the set of serviceData elements associated with a service instance a “serviceData set”. A serviceData set may also refer to the set of serviceData elements aggregated from all of the serviceData elements declared in a portType extension hierarchy (see 3.8).

Each service must convey a “logical” XML document that contains the serviceData element values, this is contained within a serviceDataValues element. An example of a serviceDataValues element is shown above. The implementation of the service is free to choose how the SDE values are stored; for example, it is not obliged to store the SDE values as XML, Java instance variables are one option.

Furthermore, the wsdl:binding associated with various operations manipulating serviceData elements will indicate the encoding of that data between service requestor and service provider. For example, a binding might indicate that the serviceData element values are encoded as serialized Java objects.

## 3.5 Operations on ServiceData

ServiceData elements can be consumed in multiple ways. There are several approaches. An approach to query, set and receive notification on serviceData value changes is described in the Grid services specification.

Once serviceData has been defined, it is necessary to indicate when and where a service allows modification (setting) or query (getting) values of that serviceData element. We say a service provides a getter operation for a serviceData element if it provides one or more operations that reference the serviceData element as a part of the output of an operation. Similarly, we say a service provides a setter operation for a serviceData element if it provides one or more operations that reference the serviceData element as a part of an input operation.

Operation-level get or set of serviceData can be defined using the following modified WSDL syntax:

```
<portType name="ncname">
  <operation name="ncname">+
    <input message="qname" serviceData="list-of-qnames"/>
    <output message="qname" serviceData="list-of-qnames"/>
  </operation>
</portType>
```

Note that while the summary syntax above doesn't indicate it, the usual WSDL 1.1 permutations of <input> and <output> are allowed to create the four styles of operations.

Note also that the same serviceData can be listed as an input and an output parameter of the same operation.

## 3.6 Defining Initial Values within the PortType

A portType can declare initial values for any serviceData element with mutability marked as “static” in its serviceData set, regardless of whether the serviceData element was declared locally, or in one of the portTypes it extends.

```
<wsdl:definitions xmlns:tns=”xxx” targetNamespace=”xxx”>
   <wsdl:portType name="exampleSDUse" extends “tns:otherPT”>
      <wsdl:operation name=…>
…
…
   <wsdl:serviceData name="sd1" type=”xsd:String”
                     mutability=”static” />
   <wsdl:serviceData name="sd2" type=”tns:SomeComplexType”/>

   <wsdl:staticServiceDataValues>
      <xxx:sd1>an initial value</xxx:sd1>
      <yyy:inheritedSDE>initial val</yyy:inheritedSDE>
   </wsdl:staticServiceDataValues>
…
   </wsdl:portType>
…
</wsdl:definitions>
```

It is illegal to declare an initial value for a serviceData element with mutability other than “static”.

## 3.7 Mutability

We provide a mutability attribute on the serviceData element declaration. This attribute indicates how the serviceData’s values may change over the lifetime of the instance

From the Grid services spec: (in the Grid services spec, the abbreviation SDE is often used for serviceData element)

> • mutability=”static”: this implies that the SDE value is assigned in the WSDL declaration (staticServiceDataValues) and remains that value for any instance of that portType. A “static” SDE is analogous to a class member variable in programming languages.

• mutability=”constant”: this implies that the SDE value is assigned by the Grid service instance and MUST not change during the lifetime of the Grid service instance once it is set to a value.
• mutability=”extend”: this implies that once elements are in the SDE value, they are guaranteed to be part of the SDE value for the lifetime of the Grid service. New elements can be added to the SDE value, but once these elements are added, they cannot be removed.
• mutability=”mutable”: this implies any of the elements in the SDE value MAY be removed at anytime, and others MAY be added.

Note: the functionality described here is different than the fixed and default attributes on the element definition in XML Schema. Fixed could be used to suggest a static value, but append and mutable would have to be modeled by a mutability attribute. The case where mutability = “constant” would be used is to specify a property that does not change after a value is assigned (but the value is not assigned by the service description, but rather, it must be initialized at runtime).

## *3.8 ServiceData Element Aggregation within a PortType Extension Hierarchy*

WSDL 1.2 has introduced the notion of multiple portType extension. A portType can extend 0 or more other portTypes and a service can implement 1 or more portTypes.

The serviceData set defined by the service’s interface is the set union of the serviceData elements defined by each portType in the transitive closure of portTypes implemented by the Web service. Because serviceData elements are identified by QName, the set union semantic implies that a serviceData element can appear only once in the set of serviceData elements.

Consider the following example:

```
<portType name=”pt1”>
   <serviceData name=”sd1” …/>
</portType>

<portType name=”pt2” extends “pt1”>
   <serviceData name=”sd2” …/>
</portType>

<portType name=”pt3” extends “pt1”>
   <serviceData name=”sd3” …/>
</portType>

<portType name=”pt4” extends “pt2 pt3”>
   <serviceData name=”sd4” …/>
</portType>
```

The serviceData sets defined by each portType is summarized as follows:

| **if a service implements…** | **its serviceDataSet** |
|---|---|

| | **contains…** |
|---|---|
| pt1 | sd1 |
| pt2 | sd1, sd2 |
| pt3 | sd1, sd3 |
| pt4 | sd1, sd2, sd3, sd4 |

## 3.9 *Initial Values of Static ServiceData Elements within a PortType Extension Hierarchy*

Initial values of static SDEs can be aggregated down a portType extension hierarchy. However, the cardinality (minOccurs and maxOccurs) can be problematic.

```
<portType name=”pt1”>
   <serviceData name=”sd1” minOccurs=”1” maxOccurs=”1”
                           mutability=”static”/>
   <wsdl:staticServiceDataValues>
      <sd1>1</sd1>
   </wsdl:staticServiceDataValues>

</portType>
```

A Web service instance that implements pt1 would have the value <sd1>1</sd1> for SDE named sd1.

```
<portType name=”pt2” extends “pt1”>
   <serviceData name=”sd2” minOccurs=”1” maxOccurs=”1”
                           mutability=”static”/>
   <wsdl:staticServiceDataValues>
      <sd2>2</sd2>
   </wsdl:staticServiceDataValues>
</portType>
```

A Web service instance that implements pt2 would inherit the value <sd1>1</sd1> for SDE named sd1 and would have the value <sd2>2</sd2> for the SDE named sd2.

```
<portType name=”pt3” extends “pt1”>
   <serviceData name=”sd3” minOccurs=”1” maxOccurs=”unbounded”
                           mutability=”static”/>
   <wsdl:staticServiceDataValues>
      <sd3>3a</sd3>
      <sd3>3b</sd3>
   </wsdl:staticServiceDataValues>
</portType>
```

A Web service instance that implements pt3 would have two values <sd3>3a</sd3> and <sd3>3b</sd3> for the SDE named sd3. It would of course inherit the value for the SDE named sd1.

```
<portType name=”pt4” extends “pt1”>
   <serviceData name=”sd4” minOccurs=”0” maxOccurs=”unbounded”
                           mutability=”static”/>
</portType>
```

A Web service instance that implements pt4 would inherit the value for sd1 defined in pt1, but the absence of a staticServiceDataValues element implies that there is no value for sd4 (although it is most likely that one would be defined in a portType which extends pt4).

```
<portType name=”pt5” extends “pt1”>
   <serviceData name=”sd5” minOccurs=”1” maxOccurs=”unbounded”
                           mutability=”static”/>
   </wsdl:staticServiceDataValues>
   <wsdl:staticServiceDataValues>
</portType>
```

A Web service instance that implements pt5 could not be created. Since there is no initial value for sd5, and the minOccurs value is greater than zero, an error is generated when the instance is created. PortTypes of this sort can be encountered if it is the intention of the designer to declare an “abstract” portType, wherein portTypes extending the abstract portType define concrete values for SDEs with minOccurs greater than zero.

```
<portType name=”pt6” extends “pt1”>
   </wsdl:staticServiceDataValues>
      <sd1>6</sd1>
   <wsdl:staticServiceDataValues>
</portType>
```

A Web service instance that implements pt6 could not be created. Since this portType declares an additional value for the SDE named sd1 (recall there is a value inherited from pt1) which exceeds the maxOccurs value for the SDE named sd1, an error is generated when the instance is created. PortTypes of this sort are in error and the designer should be ridiculed.

```
<portType name=”pt7” extends “pt2 pt3”>
   <serviceData name=”sd7” minOccurs=”1” maxOccurs=”1”
                           mutability=”static”/>
   <wsdl:staticServiceDataValues>
      <sd7>7</sd7>
      <sd3>7</sd3>
   </wsdl:staticServiceDataValues>
</portType>
```

A Web service instance that implements pt7 has a very interesting set of serviceData element values. First, it has a single value <sd1>1<sd1> for the SDE named sd1. Despite inheriting pt1 via pt2 and pt3, the initial values for sd1 are not repeated. The value <sd2>2</sd2> is the only value for the SDE named sd2, this is inherited from pt2. The SDE named pt3 has 3 values: <sd3>3a</sd3>,<sd3>3b</sd3> (inherited from pt3) and <sd3>7</sd3> locally defined.  And finally, there is a locally defined value for the SDE named sd7 (<sd7>7</sd7>).

In general, values for static SDEs are aggregated down a portType extension hierarchy. If the resulting set of SDE values violates the cardinality of the SDE (the number of values is either less than the value of minOccurs, or greater than the value of maxOccurs), an error is reported when a Web service instance is created.

### *3.10 Dynamic ServiceData Elements*

Although many serviceData elements are defined in the Web service's interface definition, there are situations that surface in the Grid where serviceData elements can be added or removed dynamically to the instance. The means by which the serviceData set of an instance may be updated is implementation specific. Note, in GridService portType, there is a serviceData element named "serviceData" that lists the serviceData elements currently defined, allowing the requestor to use the subscribeServiceData operation if this the serviceDataSet changes, and findServiceData operation to determine the current serviceDataSet value.