
Skill Level: Introductory

Babu Sundaram (babu@cs.uh.edu)
Grid Specialist
Freelance Writer

15 Mar 2005

“This tutorial introduces the concepts behind the Web Services Resource Framework (WSRF). Specifically, it explains the concept of a WS-Resource and how to create, use, and manipulate one.”

Section 1. Before you start

Who should take this tutorial?

This tutorial introduces the concepts behind the Web Services Resource Framework (WSRF). Specifically, it explains the concept of a WS-Resource and how to create, use, and manipulate one.

This tutorial is for users who wish to create Web service-based applications that require the use of "stateful resources." while the most common application of this concept is grid services, the concepts in this tutorial apply to any Web-service application that requires a notion of "state," such as those that use files or databases that exist even when you’re not talking to them. (See Stateful resources for a more complete explanation.)

Because this is a conceptual tutorial, it requires no actual programming, but a general familiarity with programming concepts, such as objects and properties, is suggested, though not required. It does, however, discuss the creation of a Web Services Description Language (WSDL) file. Basic WSDL concepts will be provided, but a general knowledge of XML and of Web services will be helpful. (See Resources for an introduction to XML and Web services.)
What is this tutorial about?

This tutorial is the first in a four-part series that introduces the concepts behind the Web Services Resource Framework (WSRF). WSRF is a group of specifications that provide a standard way to interact with "stateful" resources in the essentially "stateless" environment of a Web services application. That stateful resource can be virtually anything, from a database to an electronic hamster. In fact, you can use WSRF to work with any item that can be manipulated by changing its properties.

A WS-Resource is a combination of a stateful resource, such as a database or a hardware device, and a Web service with which it interacts. This tutorial explains the process of creating these WS-Resources, and of requesting and updating the properties that define a WS-Resource's state. We will cover the following:

- An overview of the WSRF specifications
- An overview of the system used in this series of tutorials
- Basic information about WSDL
- Basic information about WS-Addressing, used to locate a WS-Resource
- Creating a WS-Resource
- Getting and setting WS-Resource properties

Note that WSRF as a specification defines the structure of a WSDL file that describes these operations. That WSDL file can then be used by implementations in any language. This tutorial describes the creation of the WSDL file and shows the resulting SOAP messages.

It's important to understand that the WSRF specifications define what should be done, but not how it should be done. The actual implementation of the concepts described here is left to the application. In Part 4 of this series, we'll discuss this implementation using the Core Java WSRF classes provided by The Globus Alliance.

In this series, we're going to use the example of a series of satellites orbiting the Earth, taking measurements. We'll create a WS-Resource that refers to one of these satellites. We'll then request its properties to retrieve its data, change its properties to move it in the sky, and change what it's looking at.

Prerequisites

Because this tutorial does not involve programming, the only tool required is a text editor for creating WSDL files. We will, however, reference the WSRF-related WSDL files located at http://www.globus.org/wsrf.

See Resources to download the complete WSDL file covered in this tutorial.
Section 2. Overview

Introducing grid computing

Let's start by pointing out that the Web Services Resource Framework (WSRF) has applications far beyond the grid, but that's where WSRF was developed, so it acts as a perfect example of what WSRF can do. To that end, it pays to take a moment to explain just what grid is for those who may not be familiar.

Millions of Web users have already experienced grid computing, most without being aware of it. In 1998, probably the best-known grid computing project was created when the Search for Extraterrestrial Intelligence (SETI) project, reeling from its abandonment by NASA, launched a new plan. They had an unimaginable amount of data pouring out of the Aricebo radio telescope, but not enough processing power to handle it. To solve the problem, they created a screensaver that would, when the computer wasn't busy, request a packet of data from a central server, analyze it, and send back the results. It would then request a new packet and start all over again. Millions of users downloaded the screensaver and participated in the project.

Larger grid applications would be more complex and have more issues to be addressed relating to authentication, process-to-process communication across organizational boundaries, and high-performance data transport. And, while grid enthusiasts are probably sick of hearing about SETI@Home, it does provide a good example of how a grid works. It involves the following steps:

1. Work is broken down into "units" suitable for processing on multiple systems. This "work" could be computational, storage, or other another type of processing.

2. Units are distributed to multiple clients. This might be accomplished by having clients request, or "pull" work, or by having a centralized server "push" the work to available clients.

3. Overall progress, needs, and status are maintained by a centralized system or group of systems.

Current grid applications follow much of this model, with a centralized system that interacts with clients on systems that are typically geographically separated from the central server.

Using Web services: promise and problems

Now, with mutliple clients in various places, it would seem that Web services would
be a natural choice for grid computing. After all, it provides a standard, easy way to
get information from one system to another, without having to resort to platform or
language-specific methods, such as CORBA, DCOM, or Java-RMI.

That was not, however, how it happened. Early grid applications used other, less
portable methods. But why?

Perhaps the most plausible reason was an architectural one. Although a grid
application may be implemented on a number of machines, it's still a single
application, and that can be difficult to reconcile with an architecture that is,
essentially, stateless. When you connect to a database using a database client, you
remain connected; you can insert a record, then view the results of that insertion.
Another client viewing the table won't see the record unless you've committed the
transaction, but the database recognizes your session and knows it's you.

Web services don't work that way. With a Web service, you make a request (such as
inserting a record) and get a response (such as the fact that it was successfully
inserted), then you disconnect. There is no ongoing session to manage. Like HTTP
-- and in most cases, Web services travel over HTTP -- each request is independent
of the previous request, so Web services have no access to or use of information
that is not part of the current input message.

WSRF aims to solve that problem by creating the notion of "state" and a way to
manipulate it.

Stateful resources

So what, exactly, is "state," and what is a "stateful resource?"

A stateful resource is something that exists even when you're not interacting with it.
A database exists, for example, even when you're not querying it. A satellite in orbit
around the planet exists even when you're not talking to it. Even a simple counter
must exist between invocations, or every time you called it, it'd return a value of 1.

What's more, it's important to understand that the notion of state also includes the
idea of properties. When you are asked to return an object in the same state in
which you borrowed it, that involves the values of certain properties, such as the
level of cleanliness, the need for repair, or the amount of fuel in the tank. Stateful
resources are the same way; they have properties that define their state, and these
properties are how we'll interact with them, as shown in WS-Resource properties.

Web services + stateful resource = WS-Resource

According to the specification, a WS-Resource is the combination of a Web service
and a stateful resource on which it acts. But what does that actually mean?

Let's look at it from a practical matter. Suppose we have a system that involves the
management of a group of satellites. Each satellite is a stateful resource in that it
exists even when we’re not talking to it. We might also have a Web service that provides satellite functions such as changing orientation, retrieving information, or even adjusting altitude.

By combining the two, we create a WS-Resource. Note that this is not necessarily a one-to-one correspondence. For example, this single Web service might interact with several different satellites, creating several different WS-Resources that refer to the same service. On the other hand, a single satellite might interact with several different services, such as those that control on-board experiments and those that fire laser beams at incoming aliens, creating several different WS-Resources that refer to the same stateful resource.

In order to use a WS-Resource, you have to understand its properties.

**WS-Resource properties**

As mentioned in [Stateful resources](#), a stateful resource (and, therefore, a WS-Resource) has various properties associated with it. A satellite, for example, might have the following properties:

- latitude
- longitude
- altitude
- pitch
- yaw
- roll
- focalLength
- currentView

In this case, the first three properties, latitude, longitude, and altitude, specify the satellite's position. The second, pitch, yaw, and roll, specify its orientation, or the direction in which it's looking. The final properties, focalLength and currentView, specify the distance to the point on which it's focused and what it sees at that point.

The values of these properties define the state of the resource. Change a property value, and you change the state. And that, in fact, is how we're going to work with the satellite. To change its position, we'll alter one of its positional properties. To change its orientation, we'll change one of those properties. In fact, anything we want to do to this satellite (within this rather limited implementation) can be done by simply changing these properties.

But how do we do *that*?
Enter WSRF

All right, so a WS-Resource is a combination of a stateful resource and a Web service, and we manipulate it by requesting and setting its properties. So how do we do that?

There are any number of ways to do that, and that's part of the problem. What was needed was a standard way of making requests to get and set various properties.

Enter the Web Services Resource Framework (WSRF). WSRF is actually a series of specifications that define standard "message patterns," or ways to request the value of a property or properties or to specify that those properties should be altered. In fact, WSRF defines standard ways to deal with various aspects of dealing with WS-Resources, from working with their properties to grouping them together for purposes such as authentication to making sure they're destroyed in a timely manner.

The way WSRF defines these operations is by specifying how they should appear in a Web Services Description Language (WSDL) file. A WSDL file defines the messages that pass between the two sides of a Web services conversation, so by defining the WSDL file, WSRF defines the form of any interactions that take place.

When we say "WSRF," we're actually referring to several different specifications:

- **WS-ResourceProperties (WSRF-RP)** specifies the form in which ResourceProperties are defined in a WSDL file. It also specifies the form of messages that request and receive the values of these properties, and explains how to change, add, and remove properties from a WS-Resource.
- **WS-ResourceLifetime (WSRF-RL)** talks about the situation in which a WS-Resource needs to expire or be explicitly destroyed when it's no longer needed.
- **WS-ServiceGroup (WSRF-SG)** defines a way to create a collection of Web services, such as a registry of available services.
- **WS-Base Faults (WSRF-BF)** defines a standard way of indicating errors in a WSRF-based application.

You'll note that there is no entry for a specification that simply describes how all of this should work together.

In addition to providing an introduction to some of the concepts covered in the white paper, this tutorial discusses the WS-ResourceProperties specification. Later parts of this series will discuss the other WSRF specifications, as well as the Web Services Notification (WSN) specifications, referenced throughout WSRF.
Section 3. What you need to know about WSDL

What is WSDL, and why do I care?

Before moving on to the actual business of creating WS-Resources in a WSDL file, let's take a moment to look at the purpose and structure of the WSDL file in which these descriptions will reside.

Web services -- or at least the Web services relevant to WS-Resources -- consist of SOAP messages. A SOAP message has a standard "envelope" that contains a "payload." That payload is the data that gets analyzed by the server in the case of a request and the client in the case of a response. Consider this SOAP message:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <SetAltitudeRequest xmlns=
"http://example.com/satellite.xsd">
      <altitude>47700</altitude>
    </SetAltitudeRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

It contains the standard envelope, in the http://schemas.xmlsoap.org/soap/envelope/ namespace (SOAP-ENV) and the payload, in the http://example.com/satellite.xsd namespace.

The payload can be anything, and therein lies the problem: How do you define what your application expects to see, and what it will return? That's where the WSDL file comes in. Ultimately, we'll use a WSDL file to define the "message patterns" used by a WS-Resource, but in this section, we'll just look at how the parts of a WSDL file fit together.

Messages and types

Let's start by defining the actual messages we're going to send:

```
<?xml version="1.0"?>
<definitions name="Satellite"
  targetNamespace="http://example.com/satellite.wsdl"
  xmlns:tns="http://example.com/satellite.wsdl"
  xmlns:satTypes="http://example.com/satellite.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"/>
<types>
  <schema targetNamespace="http://example.com/satellite.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="SetAltitudeRequest">
      <complexType>
```

Using WS-ResourceProperties

© Copyright IBM Corporation 1994, 2008. All rights reserved.
Starting at the bottom, we've defined two types of messages. The first, `SetAltitudeInput`, is the message we're sending to the server as input. It's the message in the SOAP message you saw in What is WSDL, and why do I care? The second message, `SetAltitudeOutput`, is the response the server sends to the client. Both messages specify an element to be placed into the body of the message.

The actual definitions of those elements reside in the schema that appears at the top of the file. For example, the `SetAltitudeInput` message consists of a `SetAltitudeRequest` element, which itself contains an `altitude` element, the content of which must be a `float`.

Next, we'll combine these messages to create an operation for the server to perform.

**Port types and operations**

Now that we know what messages we're going to send, we need to specify what kind of role they'll fulfill. To do that, we'll create a `portType` and its associated operations:

```xml
<portType name="AltitudePortType">
  <operation name="SetAltitude">
    <input message="tns:SetAltitudeInput" wsa:Action="http://example.com/SetAltitude"/>
    <output message="tns:SetAltitudeOutput" wsa:Action="http://example.com/SetAltitudeResponse"/>
  </operation>
</portType>
```
Here, we've defined a portType called AltitudePortType with a single operation, SetAltitude. We could actually define any number of operations within this portType, but, for the moment, we'll keep it simple. The SetAltitude operation specifies an input message, SetAltitudeInput, and an output message, SetAltitudeOutput. (We'll deal with the wsa:Action attributes in the next section. Also, note the namespace information.)

You can also specify a fault message to be sent if there's a problem, which we'll cover later in this tutorial series, but, again, for now we'll keep it simple.

The service and binding

At this point, we've used the portType to define what can be done, but not how it should be done. To finish the process, we need to create a binding that describes how it should be done and attach it to an actual service:

Once again, let's start at the bottom, with the service element. A WSDL file can define more than one service. For example, you might have different services for different purposes, or different services with the same purpose at different locations, or different services with different bindings, such as one for SOAP, and one for SMTP.

In this case, we're defining one service, SatelliteService, with a single port, AltitudePort. But what do we know about that port? Well, we know that SOAP...
requests should be sent to http://example.com/satellite. We also know that for more information on how to send messages, we should check the AltitudeSoapBinding.

The AltitudeSoapBinding specifies that it's an implementation of the AltitudePortType, so we know what messages to send. The binding itself specifies the particulars about how the messages within each operation should be formatted. In this case, we're using the "document/literal" style, which means we're simply dropping the defined elements into the Body.

And that completes the circle. We know where we're sending messages, how to format those messages, and what those messages should be. When we get to Creating a WS-Resource, we'll look at creating the specific messages defined by WSRF, but first we need to take a detour into WS-Addressing.

---

Section 4. What you need to know about WS-Addressing

What is WS-Addressing, and why do I care?

Once upon a time, it was easy to specify the address of a Web service. All you really needed was the URL, and all of the other information was included in the SoapAction header or the message itself. Nowadays, with Web service applications getting increasingly complex, it's not always that simple. What if you wanted the reply to be sent somewhere other than the original requestor? Or what if other information, such as a session identifier, was necessary to determine the actual "location?"

Or what if you simply needed to attach to a particular instance of a Web service? That's a situation we'll come up against in the case of WS-Resources, so we'll need a way to handle it.

Enter WS-Addressing.

WS-Addressing provides a way to specify information about a location other than just a simple Universal Resource Identifier (URI) or URL. In fact, in our case, it provides a standardized way to add a good deal of information to a SOAP message. Let's construct a SOAP message, such as:

```xml
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:sat="http://example.org/satelliteSystem">
  <SOAP-ENV:Header>
    <wsa:To SOAP-ENV:mustUnderstand="1">http://example.com/satellite</wsa:To>
    <sat:SatelliteId>SAT9928</sat:SatelliteId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
It may not seem important to have this information in the actual SOAP message, but remember that the message may pass through more than one system, and even more than one transport on its way to its final destination.

To specify this information, we need to create an *EndpointReference*.

### Endpoint references

WS-Addressing introduces the concept of an *EndpointReference*. The *EndpointReference* is a way to specify the information needed to get a message to the right place with the right associated information. For example, the *EndpointReference* for the message we specified in the previous section would be:

```xml
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/02/addressing" xmlns:sat="http://example.org/satelliteSystem">
  <wsa:Address>http://example.com/satellite</wsa:Address>
  <wsa:ReferenceProperties>
    <sat:SatelliteId>SAT9928</sat:SatelliteId>
  </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

It's important to understand the relationship between an *EndpointReference* and the SOAP message because an *EndpointReference* is how we specify the location of a particular WS-Resource. For example, when we request the creation of a new WS-Resource, the response consists of an *EndpointReference* pointing to it, as you'll see.

### Section 5. Creating a WS-Resource

#### What is a WS-Resource?

At this point, you should have a good idea of what a WS-Resource is from conceptual standpoint, and you should have the basic prerequisites of WSDL and WS-Addressing pretty well in hand. Now it's time to start actually creating and using WS-Resources.

Let's start by defining just what a WS-Resource actually is. In the case of our satellite example, we might have several types of WS-Resources, such as:
• A service that sets or retrieves the altitude of a particular satellite
• A service that sets or retrieves the location or orientation of a particular satellite
• A service that provides access to a process that counts the stars in the field of view of a particular satellite

It's important to notice two things about this list: that all three WS-Resources can refer to the same satellite, and that the WS-Resource is defined by the combination of the service and the stateful resource (in this case, the satellite) and not by the number of operations it can perform.

The ResourceProperties document

Now, we can say that a WS-Resource is the combination of a Web service and a stateful resource, but how do we represent that stateful resource within an application? The answer lies in its ResourceProperties. As mentioned in Stateful resources, the state of an object can be determined by the values of its various properties. Because it's the state of the object in which we're actually interested, we can represent the stateful resource as an XML document that shows its properties. The document is called the resource properties document. In the case of our satellite, it might be something along the lines of:

```xml
<satProp:GenericSatelliteProperties
xmlns:satProp="http://example.com/satellite">
    <satProp:latitude>30.3</satProp:latitude>
    <satProp:longitude>223.2</satProp:longitude>
    <satProp:altitude>47700</satProp:altitude>
    <satProp:pitch>49</satProp:pitch>
    <satProp:yaw>0</satProp:yaw>
    <satProp:roll>32</satProp:roll>
    <satProp:focalLength>21999992</satProp:focalLength>
    http://example.com/satellite/2239992333.zip
</satProp:GenericSatelliteProperties>
```

A change in state requires a change in one or more of these values, and vice-versa.

Extending a WS-Resource

Just as we can extend a class by adding members or methods, we can extend a WS-Resource by adding properties. For example, consider the situation in which we have a satellite that also acts as a star counter. In addition to the normal properties of the stateful resource, we might have a currentCount property:

```xml
<satProp:GenericSatelliteProperties
xmlns:satProp="http://example.com/satellite"
xmlns:counterProp="http://example.com/satellite/CounterSatelliteProperties">
    <satProp:latitude>30.3</satProp:latitude>
    <satProp:longitude>223.2</satProp:longitude>
    <satProp:altitude>47700</satProp:altitude>
</satProp:GenericSatelliteProperties>
```
Notice the new information is in a separate namespace.

**Merging the WS- and the Resource: the WSDL file**

So far, we've created the representation of the stateful resource (the satellite), but to actually create the WS-Resource, we have to tie it to the service using the WSDL file.

Let's start with the basic WSDL file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Satellite"
    targetNamespace="http://example.com/satellite"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://example.com/satellite"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema targetNamespace="http://example.com/satellite"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
                schemaLocation="WS-Addressing.xsd" />
        </xsd:schema>
    </types>
</definitions>
```

The file is pretty empty at the moment, but notice that two files must be imported for all of this to work. Typical versions of the WS-ResourceProperties.wsdl and WS-Addressing.xsd files reference directories you might not have created on your machine, so, for the sake of simplicity, you can download simplified versions from the tutorial Resources.

Now that we have the framework, let's start filling it out.
Adding the resource to the WSDL

First, let's add the actual stateful resource to the file and associate it with the Web service:

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

targetNamespace="http://example.com/satellite"
xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:tns="http://example.com/satellite"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-
WS-ResourceProperties-1.2-draft-01.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

<wsdl:import namespace="http://docs.oasis-open.org/wsrf/2004/06/wsrp-
WS-ResourceProperties-1.2-draft-01.wsdl" location="WS-ResourceProperties.wsdl" />

<types>
<xsd:schema targetNamespace="http://example.com/satellite"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="latitude" type="xsd:float" />
<xsd:element name="longitude" type="xsd:float" />
<xsd:element name="altitude" type="xsd:float" />
<xsd:element name="pitch" type="xsd:float" />
<xsd:element name="yaw" type="xsd:float" />
<xsd:element name="roll" type="xsd:float" />
<xsd:element name="focalLength" type="xsd:float" />
<xsd:element name="currentView" type="xsd:string" />

<xsd:element name="GenericSatelliteProperties">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="latitude" minOccurs="1" maxOccurs="1" />
<xsd:element ref="longitude" minOccurs="1" maxOccurs="1" />
<xsd:element ref="altitude" minOccurs="1" maxOccurs="1" />
<xsd:element ref="pitch" minOccurs="1" maxOccurs="1" />
<xsd:element ref="yaw" minOccurs="1" maxOccurs="1" />
<xsd:element ref="roll" minOccurs="1" maxOccurs="1" />
<xsd:element ref="focalLength" minOccurs="1" maxOccurs="1" />
<xsd:element ref="currentView" minOccurs="1" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</types>

<portType name="SatellitePortType"
wsrp:ResourceProperties="tns:
GenericSatelliteProperties"/>
```
We've started by adding the very basics of a Web service, the actual service element and a binding that associates it with a portType. The portType itself doesn't have any operations in it yet, but the important part is the wsrp:ResourceProperties attribute. That attribute specifies that any operations the Web service performs are performed on a particular type of stateful resource, as defined by a GenericSatelliteProperties element. The GenericSatelliteProperties element is defined in the schema. The combination of that stateful resource and that Web service is the WS-Resource.

Note that the specification says that when creating a resource properties document (such as GenericSatelliteProperties in this case), you must use the style shown here, in which the original elements are defined and referenced, rather than defining them inline.

Now let's add some actual operations to the WSDL file to see how it all works.

**Requesting a new satellite**

Of course, the whole purpose of the exercise is to actually do something with the WS-Resource, so the first thing we'll do is create a reference to an actual WS-Resource instance:
At first glance, this doesn't look much different from the WSDL file we created in What you need to know about WSDL . We have a service that points to a binding that explains how to implement a portType. The portType defines one operation, createSatellite, which uses an input and an output message. Those messages are defined in the schema.

But there is one thing about this file that's a bit different from our original: Instead of returning a simple value, the service returns an EndpointReference that points to the newly created WS-Resource. Let's see how that works out in the SOAP messages.
The SOAP request

The actual SOAP request to create the WS-Resource is very simple:

```xml
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <createSatellite xmlns="http://example.com/satellite"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We don't have an actual object yet, so the request goes to the URI listed in the WSDL file, and we've defined the request as a simple `createSatellite` element.

The response is a bit more interesting.

The SOAP response

Once you send the request for a new satellite, the server creates a reference to the new WS-Resource and sends it back in the form of an `EndpointReference`:

```xml
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/02/addressing"
                          xmlns:sat="http://example.org/satelliteSystem">
      <wsa:Address>http://example.com/satellite</wsa:Address>
      <wsa:ReferenceProperties>
        <sat:SatelliteId>SAT9928</sat:SatelliteId>
      </wsa:ReferenceProperties>
    </wsa:EndpointReference>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Notice that the `Address` element of the `EndpointReference` points to the same URI we have listed in the WSDL file, so the information is still going to the same place, there's just more of it.

Now, we should note here that this is no ordinary endpoint reference. The `ReferenceProperties` element shows an identifier that will ultimately be used to identify the WS-Resource, so this is actually a `WS-Resource qualified endpoint reference`. We can use this information to make subsequent calls to the WS-Resource, as you'll see.
Section 6. Working with a WS-Resource: Getting properties

What we're trying to accomplish

All right, we've created the WS-Resource, now what can we do with it?

Just about anything we can accomplish by adjusting its properties, in fact. For example, we could change the size of the satellite's orbit by changing its altitude property. No, just changing the value doesn't move the satellite; the actual moving of the satellite is up to the application behind the Web service. But that's really what WS-Resources are about: creating a way to manipulate objects by changing their properties. The specification just tells you how to tell the Web service about those changes. It doesn't care how the application actually manipulates the object, and neither do we.

But before we start actually changing properties, let's take a look at them. In this section, we'll first look at the altitude property for the satellite WS-Resource we created in Creating a WS-Resource. We'll then move on to requesting all of the orientation values in one go. And then we'll look at using XPath to query across multiple values.

Requesting a property

Requesting the value of a property is a simple matter of constructing the appropriate SOAP message. For example, suppose we wanted to request the value of the altitude property. The basic SOAP message would look something like this:

```xml
 xmlns:wsrp=""http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"">  
   <SOAP-ENV:Header>...</SOAP-ENV:Header>  
   <SOAP-ENV:Body>  
      <wsrp:GetResourceProperty xmlns:satProp="http://example.com/satellite">  
         satProp:altitude  
      </wsrp:GetResourceProperty>  
   </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

The `wsrp:GetResourceProperty` element is part of the WS-ResourceProperties specification. We don't even have to define it in the WSDL file. It gives us a place to specify the property for which we want to get back the value.

The SOAP message, however, isn't actually complete yet. Yes, it's a SOAP
message, but if we sent it to the Web service like this, the service wouldn’t know what WS-Resource we were referring to. We’ll take care of that next.

The complete SOAP request

In the previous section, Requesting a property, we created a SOAP message that specifies the property we want to retrieve, but for which WS-Resource?

When we created the satellite, the Web services returned an endpoint reference that pointed to the newly created WS-Resource. We can take that information and add it to the Header of the SOAP message, like so:

```xml
    xmlns:wsa="http://www.w3.org/2005/02/addressing"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
    <SOAP-ENV:Header>
        <wsa:Action>
        </wsa:Action>
        <wsa:To SOAP-ENV:mustUnderstand="1">
            http://example.com/satellite
        </wsa:To>
        <sat:SatelliteId>SAT9928</sat:SatelliteId>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <wsrp:GetResourceProperty xmlns:satProp="http://example.com/satellite">
            satProp:altitude
        </wsrp:GetResourceProperty>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `wsa:Action` element isn’t part of the original endpoint reference; it changes depending on what it is we’re trying to do. In this case, we’re using the `GetResourceProperty` action. The `wsa:To` element takes the value from the `wsa:Address` in the endpoint reference, and any `wsa:ReferenceProperty` values are included directly in the `Header`.

You’ll notice that we haven’t discussed the `SatelliteId` value. That’s intentional. Any information included in the endpoint reference for the purpose of identifying the specific WS-Resource must be ignored by your application, except to pass it on when sending messages. According to the specification, even trying to interpret the value is considered improper. It’s meant to be passed on as “black box,” leading an unexamined life.

Receiving a ResourceProperty

Once you request a property, you need to get the value back, and the WS-ResourceProperties specification also defines the form of that message. In our case, we’d receive a message such as:
In this case, the Header information refers not to the service but to the client; http://example.com/myClient is the URI of the client that should receive the response. In most cases, that’s the same as the client that made the request, but you can use the wsa:Reply-To element to send the response elsewhere.

As far as the actual body of the message, where we again have a standard element, wsrp:GetResourcePropertyResponse, but in this case it contains the actual property requested, along with its current value.

Now let’s see how this looks in the WSDL file.

wsrp:GetResourcePropertyResponse: The WSDL file

In order to add these capabilities to the application, we’ll need to add them to the WSDL file, but because we’re using standard message exchange patterns that have already been defined, we only have to add a new operation, like so:

```xml
<definitions name="Satellite" targetNamespace="http://example.com/satellite"
xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:tns="http://example.com/satellite"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/03/soap/
xmlns:wspwsdl="http://docs.oasis-open.org/wsrp/2004/06/wsrp-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
>
<wsp:import namespace="http://docs.oasis-open.org/wsrp/2004/06/wsrp-WS-
ResourceProperties-1.2-draft-01.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/>
</definitions>
```
Starting with the portType, we're creating a new operation called getAltitude. That operation has an input and an output message, but both messages have
already been defined in the WS-ReferenceProperties.wsdl file we imported earlier, so all we have to do is reference them using the appropriate namespace alias.

Once we’ve created the operation, we simply need to add it to the binding, and we’re good to go.

Requesting multiple properties

Fortunately, we’re not limited to retrieving single property values. We can also retrieve multiple properties:

```xml
   xmlns:sat="http://example.org/satelliteSystem"
   xmlns:wsa="http://www.w3.org/2005/02/addressing"
   xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-Resour
ceProperties-1.2-draft-01.xsd">
   <SOAP-ENV:Header>
      <wsa:Action>
         http://docs.oasis-open.org/wsrf/2004/06/
         WS-ResourceProperties/GetMultipleResourceProperties
      </wsa:Action>
      <wsa:To SOAP-ENV:mustUnderstand="1">
         http://example.com/satellite
      </wsa:To>
      <sat:SatelliteId>SAT9928</sat:SatelliteId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <wsrp:GetMultipleResourceProperties
         xmlns:satProp="http://example.com/satellite">
      </wsrp:GetMultipleResourceProperties>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As before, the information in the Header comes from the endpoint reference.

Receiving multiple properties

The response message is similar to its single-property counterpart:

```xml
   xmlns:sat="http://example.org/satelliteSystem"
   xmlns:wsa="http://www.w3.org/2005/02/addressing"
   xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-Resour
ceProperties-1.2-draft-01.xsd">
   <SOAP-ENV:Header>
      <wsa:Action>
         http://docs.oasis-open.org/wsrf/2004/06/
      </wsa:Action>
      <wsa:To SOAP-ENV:mustUnderstand="1">
         http://example.com/satellite
      </wsa:To>
   </SOAP-ENV:Header>
</SOAP-ENV:Envelope>
```
Again, let's add this to the WSDL file.

The WSDL file

Once again, because the actual messages we're passing have been defined in the WS-ResourceProperties.wsdl file, all that is needed to add this capability to the application is to create a new operation:

```xml
...<message name="CreateSatelliteRequest">
  <part name="request" element="tns:createSatellite"/>
</message>

<message name="CreateSatelliteResponse">
  <part name="response" element="tns:createSatelliteResponse"/>
</message>

<portType name="SatellitePortType">
  wsrp:ResourceProperties = "tns:GenericSatelliteProperties">

  <operation name="createSatellite">
    <input message="tns:CreateSatelliteRequest" wsa:Action="http://example.com/CreateSatellite"/>
  </operation>

  <operation name="getAltitude">
  </operation>

  <operation name="getOrientation">
  </operation>
</portType>
```
Once we add the `getOrientation` operation to the binding, it's ready for use by the application.

We can also query across properties, as you'll see.

**Using an XPath Query**

Although it's fairly straightforward to request the property or properties desired, there may be situations in which you need to pursue another tactic. Rather than simply requesting a resource property by name, we can use a query such as those available in XPath. (See the Resources for more information about XPath.) For example, if we're unsure whether a particular property exists, you might want to use an XPath function on the property.

Another useful capability that comes with XPath is the ability to request more than one identically named property, as we'll do in Understanding WSRF (Part 2), or even to query across parameters, as we'll do here. For example, we could use an XPath expression, along with the `boolean()` function, to determine whether the satellite is
pointed in the proper direction without having to explicitly analyze the data:

```xml
xmlns:sat="http://example.org/satelliteSystem"
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
  <SOAP-ENV:Header>
    <wsa:Action>
    </wsa:Action>
    <wsa:To SOAP-ENV:mustUnderstand="1">
      http://example.com/satellite
    </wsa:To>
    <sat:SatelliteId>SAT9928</sat:SatelliteId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsrp:QueryResourceProperties>
      <wsrp:QueryExpression
        Dialect="http://www.w3.org/TR/1999/REC-xpath-19991116">
        boolean(/*/pitch=25 and /*/roll=0 and /*/yaw=10)
      </wsrp:QueryExpression>
    </wsrp:QueryResourceProperties>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note the use of the Dialect attribute to distinguish between XPath V1.0 (shown here) and XPath V2.0 (http://www.w3.org/TR/2003/WD-xpath20-20031112). The specification doesn't limit the dialects you can support, but if the implementation doesn't recognize the Dialect, it must return a fault, or error.

**Query results**

The results look very much like the previous two responses:

```xml
xmlns:sat="http://example.org/satelliteSystem"
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
  <SOAP-ENV:Header>
    <wsa:Action>
    </wsa:Action>
    <wsa:To SOAP-ENV:mustUnderstand="1">
      http://example.com/myClient
    </wsa:To>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <wsrp:QueryResourcePropertiesResponse>
      false
    </wsrp:QueryResourcePropertiesResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
In this case, we're just returning a boolean value, but you can return any type of value XPath can return.

**XPath: The WSDL file**

Once again, we add a new operation to the WSDL file:

```xml
...<portType name="SatellitePortType"
    wsrp:ResourceProperties="tns:GenericSatelliteProperties">

...<operation name="getOrientation">
    <input message="wsrpwsdl:GetMultipleResourcePropertiesRequest"
    <output message="wsrpwsdl:GetMultipleResourcePropertiesResponse"
</operation>

<operation name="checkOrientation">
    <input message="wsrpwsdl:QueryResourcePropertiesRequest"
    <output message="wsrpwsdl:QueryResourcePropertiesResponse"
</operation>

</portType>

</binding>

<service name="SatelliteService">
    <port name="SatellitePort" binding="tns:SatelliteSoapBinding">
        <soap:address location="developerWorks® ibm.com/developerWorks"/>
    </port>
</service>
```
That takes care of retrieving properties. Now let's look at setting their values.

Section 7. Working with a WS-Resource: Setting properties

What we're trying to do

So far, we've created a WS-Resource, and we've looked at one or more properties that represent its state. That doesn't do much for actually manipulating the WS-Resource, however. In this section, we're going to look at adding, hanging, and deleting properties of a WS-Resource.

So far, our satellites have remained stationary in the sky, in a geosynchronous orbit, not looking at anything in particular. In this section, we'll add a ResourceProperty that represents a particular target. We'll then move the satellite toward that target by updating the position properties, then we'll delete the target property we added. Finally, we'll put it all together by adding the appropriate operations to the WSDL file.

Adding a property

Adding a property to a WS-Resource involves using the Insert element:

```xml
xmlns:sat="http://example.org/satelliteSystem"
xmlns:wsa="http://www.w3.org/2005/02/addressing"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.xsd">
  <SOAP-ENV:Header>
sourceProperties</wsa:Action>
sourceProperties</wsa:To>
    <sat:SatelliteId>SAT9928</sat:SatelliteId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
      <wsrp:Insert>
```
We can call the new property pretty much anything, but we'll have to allow for this new element in the Resource Properties document. (We'll do that when we adjust the WSDL file in the WSDL file.)

The result of adding a property

When we successfully add, remove, or change a property, we get a response message that simply acknowledges the action:

This response is identical to the one produced by the Update and Delete actions.

Changing a property value

In Adding a property, we added a new property, but we can also change the values of existing properties. For example, we can tell the system to move the satellite by altering its position properties using the Update element:
In this case, we're using two Update components, but we can actually use any combination of the Insert, Update, and Delete components.

Removing a property

A property can be removed completely. For example, if we've decided not to look at a particular target anymore, we can remove that property:

Again, it's imperative that the schema definition for our resource property document allows us to make this change.

Resource property document: The WSDL file
Once again, because the standard elements have already been defined within WS-ResourceProperties.wsdl, we can simply add the new operations:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Satellite"
    targetNamespace="http://example.com/satellite"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://example.com/satellite"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsi="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wspwsdl="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
        location="WS-ResourceProperties.wsdl" />
<definitions name="Satellite"
    targetNamespace="http://example.com/satellite"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://example.com/satellite"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsi="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsp="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wspwsdl="http://docs.oasis-open.org/wsrf/2004/06/wsrp-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    location="WS-ResourceProperties.wsdl" />
<xsd:schema targetNamespace="http://example.com/satellite"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="WS-Addressing.xsd" />
<xsd:element name="createSatellite">
    <xsd:complexType/>
</xsd:element>
<xsd:element name="createSatelliteResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="wsa:EndpointReference"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="latitude" type="xsd:float"/>
<xsd:element name="longitude" type="xsd:float"/>
<xsd:element name="altitude" type="xsd:float"/>
<xsd:element name="pitch" type="xsd:float"/>
<xsd:element name="yaw" type="xsd:float"/>
<xsd:element name="roll" type="xsd:float"/>
<xsd:element name="focalLength" type="xsd:float"/>
<xsd:element name="currentView" type="xsd:string"/>
<xsd:element name="GenericSatelliteProperties">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="latitude" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="longitude" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="altitude" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="pitch" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="yaw" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="roll" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="focalLength" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
</definitions>
```
<xsd:any/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
<message name="CreateSatelliteRequest">
  <part name="request" element="tns:createSatellite"/>
</message>
<message name="CreateSatelliteResponse">
  <part name="response" element="tns:createSatelliteResponse"/>
</message>
<portType name="SatellitePortType" wsrp:ResourceProperties="tns:GenericSatelliteProperties">
  <operation name="createSatellite">
    <input message="tns:CreateSatelliteRequest" wsa:Action="http://example.com/CreateSatellite"/>
  </operation>
  <operation name="getAltitude">
  </operation>
  <operation name="getOrientation">
  </operation>
  <operation name="checkOrientation">
  </operation>
  <operation name="addTarget">
  </operation>
</portType>
<output message="wsrpwsdl:
    SetResourcePropertiesResponse"
    wsa:Action="http://docs.oasis-open.org/
<operation name="moveToTarget">
    <input message="wsrpwsdl:
        SetResourcePropertiesRequest"
        wsa:Action="http://docs.oasis-open.org/
    <output message="wsrpwsdl:
        SetResourcePropertiesResponse"
        wsa:Action="http://docs.oasis-open.org/
</operation>
<operation name="removeTarget">
    <input message="wsrpwsdl:
        SetResourcePropertiesRequest"
        wsa:Action="http://docs.oasis-open.org/
    <output message="wsrpwsdl:
        SetResourcePropertiesResponse"
        wsa:Action="http://docs.oasis-open.org/
</operation>
</portType>

<binding name="SatelliteSoapBinding"
type="tns:SatellitePortType">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/
        soap/http" />
    <operation name="createSatellite">
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="getAltitude">
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="getOrientation">
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="checkOrientation">
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    <operation name="addTarget">
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
Notice also that we can add arbitrary elements to the GenericSatelliteProperties so that we can easily add a new property, such as targetCoords.

Section 8. Summary

In this tutorial, the first in a series of four on WSRF, we started by explaining the purpose behind WSRF, and why Web services alone weren't enough. We then explained that a WS-Resource is the combination of a stateful resource, such as a database or satellite with a Web service.

The resource itself is described by a series of properties, which are associated with the Web services in the WSDL file. We also covered the basics of WSDL and WS-Addressing, which WSRF uses to point to a particular WS-Resource instance.

We covered creating WS-Resources, looking at their properties, and adjusting those properties, in order to manipulate the resource.

In future parts of this tutorial series, we'll look at more advanced uses of WSRF, such as ServiceGroups and fault handling, as well as WS-Notification. In the final part of this series, we'll put it all together and write an application that uses classes implementing each of the concepts discussed in the first two parts of the series.
Resources

Learn

• The main location for WSRF documentation is at the Globus Alliance, but the latest specifications can be found at OASIS.
• Check out The WS-Resource Framework at Globus.org.
• Learn more about Web Services Resource Properties 1.2 at OASIS.org.
• WS-ResourceLifetime (WSRF-RL)
• WS-ServiceGroup (WSRF-SG)
• WS-Base Faults (WSRF-BF)
• The actual recommendations for SOAP and WSDL are maintained by the World Wide Web Consortium, and IBM is one of the companies that proposed WS-Addressing.
• Read "Merging grids and Web services" for more information.
• Be sure to read "Using WSDL in SOAP applications."
• "Deploying Web services with WSDL: Part 1" offers an introduction to Web services.
• Read "Discover SOAP encoding's impact on Web service performance" to improve performance by changing your encoding style.
• "The hidden impact of WS-Addressing on SOAP" looks at the current state of the "Web services revolution."
• You can find more information on the developerWorks SOA and Web services zone.
• The XML and XPath V1.0, XPath V2.0, and XML Schema recommendations are maintained by the World Wide Web Consortium.
• Read "A survey of XML standards."
• developerWorks offers tutorials titled "Introduction to XML" and "Get started with XPath."
• You can also find more resources in the developerWorks XML zone.

Get products and technologies

• Download part1wsrf.wsdl for the complete WSDL file covered in this tutorial.

About the author

Babu Sundaram
Babu Sundaram has been actively involved with grid computing research since the
early days of Globus Toolkit. He was part of the Globus implementation team during his internship at Argonne National Labs. He holds a bachelor's degree in mechanical engineering and master's degree in computer science. He is pursuing his doctoral-level research in computer science at the University of Houston. He has published many papers in various conferences and grid-related workshops, and also co-authored sections of a few books about grid computing. He loves teaching and, at times, works as lecturer teaching courses on Web services and aspects of grid computing. He welcomes feedback and can be reached at babu@cs.uh.edu.