Implement a publish-subscribe WSRF NewsFeed service

Create a simple Globus 4 grid service that leverages WS-Notification

Skill Level: Intermediate

Duane Merrill (duane@merrill.com)
Developer
Ph.D. Candidate

15 Nov 2005

The WS-Notification family of specifications describe a system where users can subscribe for notifications on various topics, and editors can approve and send messages. It does not, however, explain how to implement such a system. This tutorial shows how to create such an application using the Globus WSRF Core classes.

Section 1. Before you start

This tutorial illustrates how to develop a Globus Toolkit V4 (GT4) grid service that makes use of the WS-Notification publish-subscribe pattern for communication. The tutorial is written for Web service and grid developers who would like a blueprint for quickly prototyping a grid service that incorporates WS-Notification from within the Eclipse IDE. A general familiarity with Java™ technology, Eclipse, Web services, the Web Services Resource Framework (WSRF), WS-Notification (WSN), and Globus Toolkit V4 (GT4) is recommended.

About this tutorial

Our example grid service, NewsFeed, operates in a publish-subscribe fashion by
accepting news articles supplied by reporters and forwarding them to editors subscribed to their relevant subjects. It is implemented as a GT4 grid service and follows the WSN standard of publish-subscribe behavior.

In this tutorial, we will cover the following:

- A brief overview of the Web services technologies used by the example NewsFeed service
- Creating the initial GT4 Eclipse project
- Modifying and reviewing the project source files (the sources for these project files are all provided in the Resources section)
- Using the GT4IDE Eclipse plug-in to facilitate the automatic generation of the remaining artifacts, assembling the grid archive (GAR), and deployment of the grid service into the Web services container
- Running and debugging the grid service

Prerequisites

To run the example from this tutorial, you'll need to obtain and install the following components and tools. (see Setting up):

- Java SDK V1.4.2
- Eclipse IDE V3.0.2
- Apache Jakarta Tomcat V5.0
- Sysdeo Eclipse Tomcat Launcher Plug-in V3.1 (beta at the time of this writing)
- GT4 WS Core V4.0.1
- GT4IDE Eclipse plug-in 0.1-a3

Section 2. Grid computing with the Web service paradigm

This section provides an overview of the relationships between the among
participants and the fundamental specifications in the Web services grid computing field, and reviews the fundamentals of the Web Services Resource Framework (WSRF) and WS-Notification (WSN) families of specifications.

The ideal situation

The appeal of the Web services model is largely due to its emphasis on interoperability and its loosely coupled nature, both of which make it an attractive technology for the newest generation of Internet-based service-oriented distributed computing infrastructures. There are a number of communities, organizations, and working groups devoted to standardizing this new paradigm of Web services-based grid computing, and it’s easy to get lost in the alphabet soup:

- **The Global Grid Forum (GGF)** encapsulates a variety of community-initiated working groups developing grid computing best practices and specifications in cooperation with other leading standards organizations, software vendors, and users.

- **The Open Grid Services Architecture (OGSA)** is aimed at identifying and standardizing (by specifying interfaces for) the services that one commonly finds in a grid application. For example, a typical grid application could comprise job management services, resource management services, security services, etc. -- all defined by OGSA. OGSA is a product of the grid community at large, but specifically attended to by the GGF.

- **Organization for the Advancement of Structured Information Standards (OASIS)** is an international not-for-profit consortium that drives the development, convergence, and adoption of e-business standards. OASIS is the primary producer of today’s Web services standards. Both WSRF and WSN are OASIS standards.

- **The Web Services Resource Framework (WSRF)** is a family of specifications developed by OASIS to ease the burden of implementing and maintaining complex distributed systems by providing standard ways to interact with OGSA components. WSRF specifies how one invokes the interfaces established in the OGSA, and it provides the specification for the stateful Web services OGSA needs. We will explore the nature of stateful WSRF-compliant services later.

- **WS-Notification (WSN)** is an OASIS-sponsored specification that defines a standard Web services approach for implementing the publish-subscribe pattern of communication. We will explore WSN later.
• **The Globus Toolkit (GT)** is a software tool kit that is a WSRF-compliant set of software components from which developers can build distributed systems. GT4 provides many of the services required by the OGSA. Because of The Globus Alliance's depth of experience in the grid and distributed computing fields, and the widespread use of previous versions of the toolkit, GT4 is poised to be one of the premier enabling technologies for grid services and applications as Web services-based grid and distributed computing standards mature.

**WSRF**

Web services comprise a terrific medium for implementing Internet-based distributed services. However, plain vanilla Web services are generally stateless, meaning that they do not maintain state between invocations. Unlike more tightly coupled technologies like Enterprise JavaBeans (EJB), Web services do not have provisions for persisting information between interactions. This presents a problem for implementing a variety of grid components as Web services. Consider our example NewsFeed grid service in this tutorial: It would not function very well if an editor's subscription to a certain class of articles only exists during the process of making that subscription.

WSRF addresses this by introducing the notion of a WS-Resource, which is the pairing of a Web service that exposes one or more stateful resources.

**Figure 1. A WS-Resource**

WS-Resource pairings (WS-Resources) are addressed by end-point references (EPRs). The EPR is defined in the WS-Addressing specification from OASIS, and contains (among other things) a URI for identifying the Web services component of WS-Resources and a resource key to identify the resource to manipulate.
The state for WS-Resources is maintained as a group of properties called ResourceProperties. The WS-ResourceProperties document is a WSRF specification that outlines the get, set, and query port types that allow us to interact with ResourceProperties.

WSN

The publish-subscribe (also known as event-driven, notification-based, or observer/observable) design pattern is a popular approach to communication used in loosely coupled systems. In this pattern, a producer entity disseminates information (usually in the form of one-way messages) to a set of consumer entities that have established interest (a subscription) for that information. It fits scenarios in which consumer entities wish to stay informed regarding the producer entity by being notified regarding any changes, events, or situations that concern it. For example, a hospital system may want to page doctors in response to changes in their patients' conditions.

The loosely coupled nature of the Web services paradigm lends itself well to the publish-subscribe design pattern because the entities involved require very little prior information about each other to communicate. Under the publish-subscribe pattern, the producer/consumer relationship is dynamic and ad-hoc, and the producer has little or no reliance on the consumer(s).

WSN (developed in conjunction with WSRF) is an OASIS-sponsored family of related specifications that defines a standard Web services approach to notification using a topic-based publish-subscribe pattern. WSN comprises three normative specifications:

1. **WS-Tops** -- This specification establishes the notion of WS-Tops, which are items of interest for subscription. Topics are versatile, and can be organized and derived from each other hierarchically (much like event categories in Log4j), as well as aliased to each other.

2. **WS-BaseNotification** -- This specification presents the standard interfaces for consumers (NotificationConsumers) and producers (NotificationProducers), primarily focusing on the acts of subscription and notification. Much flexibility is given to the subscription process. In addition to specifying topics of interest, it facilitates pausing, resuming, and controlling the length of a subscription.

3. **WS-BrokeredNotification** -- This document concerns itself with the case in which the producer does not manage the subscriptions itself but
Section 3. Setting up

This section describes the design for the NewsFeed service example, and reviews the tools and components that we will be using to implement it.

Design overview

Our NewsFeed service will store the most recently submitted articles for five genres: weather, politics, local news, sports, and editorials. When any of these articles is replaced with a newer one, any clients that have subscribed to those topics will be notified with the newest story.

We can design our NewsFeed service as a Web service paired with one resource that maintains these five articles as ResourceProperties (see Figure 2). In GT4 terms, this means that it follows the "Singleton with ServiceResourceHome" pattern (as opposed to other patterns like the "Factory/Instance" pattern"). In fact, they will both be implemented within the same Java class.

Figure 2. Operation of the NewsFeed service
GT4 simplifies the coding for this example project drastically by providing the ability to have `ResourceProperties` automatically serve as WSN topics. This means that whenever a resource property (a sports or weather article) is updated, GT4 automatically handles the sending of notifications to subscribers of the corresponding topic.

For this tutorial, I have made a sandbox for all the sources and tools in my `\Dev\Tutorial` directory. When following along, you can replace this prefix with any reasonable substitute.

Additionally, I downloaded and installed Java SDK V1.4.2. (This tutorial was written using version 1.4.2.07.)

**Eclipse**

In this tutorial, we are going to use the Eclipse IDE to develop, deploy, and debug the NewsFeed grid service. The Eclipse tool integration platform is a popular, extensible well-documented IDE that can be configured to host all of the useful development activities from coding to deployment to debugging. We will be using it as our Java IDE.
The Eclipse Platform distribution should be obtained and unzipped into the local file system. For this tutorial, I installed Eclipse V3.0.2 into the \Dev\Tutorial\eclipse directory. For more information about Eclipse, see Resources.

**GT4 WS-Core**

The GT4 WS-Core is The Globus Alliance's Java implementation of the WSRF and the WSN family of standards. It provides the API and tools for building stateful WS-Resources. We will be using the GT4 API to create our NewsFeed service and using its container logic to deploy it.

The GT4 WS-Core distribution should be obtained and unzipped to the local file system. I downloaded the V4.0.1 source-code distribution (as opposed to the binary distribution) so I could step into and view the WS-Core source while debugging and built it into \Dev\Tutorial\GTK, which is what I set my %GLOBUS_LOCATION% environment variable to. For more information about GT4, see Resources.

**GT4IDE**

Although writing the business logic GT4 WSRF grid service such as this one may be simple, there are a lot of other artifacts (such as WSDL interfaces, deployment descriptors, namespace mappings, GARs, etc.) that need to be coordinated, assembled, and deployed before we can actually run our service. Fortunately, we can use the GT4IDE Eclipse plug-in, which handily automates the process of:

- Generating the service stubs and parameter data structure classes
- Updating the service WSDL
- Assembling the service GAR (the grid archive that contains all the files and information the Web Services container needs to deploy our service)
- (Optionally) Deploying the GAR into the GT4 directory tree
- (Optionally) Deploying the GT4 WSRF into Tomcat

Because we’re developing against the JDK V1.4, I made sure to obtain the corresponding version of the GT4IDE. (A JDK V1.5 version of the GT4IDE exists, as well.) For more information about GT4IDE, see Resources.

**Sysdeo Eclipse Tomcat Launcher Plug-in**
In order to completely contain the iterative (develop/deploy/debug/) development process within Eclipse, it is convenient to have a mechanism for Eclipse to run and debug the NewsFeed as well. As a GT4 grid service, NewsFeed needs to be run within and managed by a Web services container. Although the GTK ships with its own stand-alone Web services container, we can use the popular Apache Tomcat servlet container to host our example grid service by leveraging the Sysdeo Eclipse Tomcat Launcher Plug-in to integrate Tomcat into the Eclipse environment. In addition to enabling us to debug the service from within Eclipse, the embedded Tomcat allows us to update our service logic on the fly and see the changes reflected immediately.

This aspect of the tutorial is optional; deploying and running the NewsFeed service GAR within the stand-alone GT4 container will work just fine.

The Tomcat distribution should be obtained and unzipped into the local file system. For this tutorial, I installed Tomcat V5.0.28 in the \dev\jakarta-tomcat-5.0.28 directory. For more information about Tomcat, see Resources.

The Sysdeo Eclipse Tomcat Launcher Plug-in should be obtained and unzipped into the Eclipse plug-ins directory. (Its features will be made available the next time you start Eclipse.) For more information, see Resources.

Section 4. Creating and configuring the project

This section describes how we're going to create a new Java project for our NewsFeed service and how to complete the skeleton service source files. Although this is a programming tutorial (rather than a tools tutorial), we'll cover the project-creation process in a little bit of detail here because the GT4IDE plug-in does a fantastic job of creating all the source files and skeleton implementation that we'll need.

Create a new project

We start by creating a new Java project called NewsFeed. Select File > New > Project and select GT4 Project from the selection wizard (see Figure 3). Click Next and enter NewsFeed in the Project Name text box. Check the Create First Service checkbox. Click Next to continue.
The next dialog requests the majority of the information needed to create the skeleton source files for the grid service:

**Figure 4. Entering information for service skeleton generation**
Fill in the following fields:

- **Base Package** -- This is the base Java package where the service implementation, the WSDD file, and the stub subpackages will be placed. The WSDD is the configuration file that tells the Web service container (Tomcat) how to publish the Web service. I used `org.merrill.examples.NewsFeed`.

- **Base Namespace** -- This is the base of the target namespace of the (G)WSDL file -- in this case, it's `http://examples.merrill.org/newsfeed`. The full namespace for the service is derived by combining this field with the following Name of Interface/PortType field.

- **Name of Interface/PortType** -- This is the basename part of the full service namespace. I specified `NewsFeed`. Many other names in the WSDL will be derived from this value. For example, the portType subclass will be called `NewsFeedPortType`. Hence, the namespace for the service
will be http://examples.merrill.org/newsfeed/NewsFeed.

- **Service Path** -- URL where the service will be published in Tomcat. Note that this is not a full URL (since all service URLs start with http://localhost:8080/wsrf/services). By specifying "examples," the full publication path of the service will be http://localhost:8080/wsrf/services/examples/NewsFeed.

- **Design Pattern** -- As mentioned, we're opting for the "Singleton with ServiceResourceHome" pattern. Our Web service does not need to back-end to multiple NewsFeed resources.

- **Provide implementation skeleton** -- We set this to "Yes, using ReflectionResourceProperties" to request that the plug-in generate a set of basic Java skeleton files for us to flesh out.

If you are going to be using Tomcat and Sysdeo to host the NewsFeed service, be sure to update the project properties to enable the project as a Tomcat Project.

**The project files**

It's finally time to complete the file skeletons generated by the GT4IDE plug-in. If you switch to the Resources Perspective in Eclipse, the Navigator should now look like Figure 5.

**Figure 5. Project files created by GT4IDE**
Here's a quick overview of the files generated for our NewsFeed service and what they do:

- NewsFeed.wsdl -- The XML document that describes the service interface. The service interface describes how the outside world can interact with our service -- specifically, the operations that can be performed on it. We will be modifying this WSDL to support WS-Notification and to expose our article resource properties.

- NewsFeedQNames.java -- A convenient Java interface class containing the QName URI/namespace constants relevant to our NewsFeed grid service. By having our service (and client) classes implement this
interface, we can reference these constants without having to replicate them throughout the project. We will modify this class to add the QName for our resource properties.

- NewsFeedService.java -- This is the business logic for our NewsFeed service. Because we opted for the "Singleton with ResourceHome" pattern, the NewsFeedService Java class contains the service and the resource implementations. We will modify this class to expose our new resource properties and configure them to serve as WS-Notification topics.

- build.xml -- The Ant buildfile used by GT4IDE that contains the artifact-generation for this service. The Generate GAR File button in the Globus Toolkit IDE Perspective calls the jars target. We can edit this target to do additional work, such as deploy the GAR into the GT4 tree and deploy GT4 into Tomcat.

- deploy-jndi-config.xml -- This file is the JNDI deploy file that enables the GT4 WSRF implementation to locate the resource-home for this service. The resource-home is in charge of managing the resources. We won't need to modify this file for the simple NewsFeed service because it follows the "Singleton with ResourceHome" pattern.

- deploy-server.wsdd -- This file is the WSDD configuration file that tells the Web Service container (Tomcat) how to publish the Web service. When we modify our WSDL to extend our NewsFeed service portType from the WS-ResourceProperties and WS-Notification portTypes, we'll need to modify this file to "plug in" the GT4 providers for these methods.

- namespace2package.mappings -- Because WSDL is language-neutral, this file provides a way to map namespaces to Java implementation classes. We won't need to modify this file.

### Updating the Service WSDL for WSN and ResourceProperties

The first thing we must do to the NewsFeed WSDL is to extend it to support WSN. Each portType within a WSDL effectively denotes an interface for the service. PortTypes are extensible, meaning that our NewsFeed service portType can derive from other portTypes. We need to extend our portType to expose the subscribe() operation, and we do this by deriving from the NotificationProducer WSN portType.

First, we need to add (shown in bold) the WS-Notification namespace to the definitions tag at the top of our WSDL, as shown in Listing 1.

#### Listing 1. Updates to NewsFeed.wsdl to declare the WS-Notification namespace

```xml
<definitions
  xmlns:wsn="http://schemas.xmlsoap.org/ws/2002/10/Notification" ...
```
namespace

...<definitions name="NewsFeedService" targetNamespace="http://examples.merrill.org/newsfeed/NewsFeedService"
xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:tns="http://examples.merrill.org/newsfeed/NewsFeedService"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:wsdlpp="http://www.globus.org-namespaces/2004/10/WSDLPreprocessor"
xmlns:wsl="http://docs.oasis-open.org/wsrf/2004/06/
wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/
wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/
wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
xmlns:wsntw="http://docs.oasis-open.org/wsn/2004/06/
wsn-WS-BaseNotification-1.2-draft-01.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...

We can now import the WSN WSDL into our service WSDL file, as shown in Listing 2.

Listing 2. Updates to NewsFeed.wsdl to import the WSN WSDL

...<wsdl:import
namespace= "http://docs.oasis-open.org/wsrf/2004/06/
wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
location="../wsrf/properties/WS-ResourceProperties.wsdl" />
<wsdl:import
namespace= "http://docs.oasis-open.org/wsrf/2004/06/
wsrf-WS-ResourceLifetime-1.2-draft-01.wsdl"
location="../wsrf/lifetime/WS-ResourceLifetime.wsdl" />
<wsdl:import
namespace= "http://docs.oasis-open.org/wsn/2004/06/
wsn-WS-BaseNotification-1.2-draft-01.wsdl"
location="../wsrf/notification/WS-BaseN.wsdl"/>
...

Finally, we need to update the declaration of our NewsFeedPortType to extend the ResourceProperties and WS-Notification interfaces (see Listing 3).

Listing 3. Updates to NewsFeed.wsdl

...<portType name="NewsFeedPortType"
wsdlpp:extends="wsrpw:GetResourceProperty wsrpw:GetMultipleResourceProperties
wsrpw:SetResourceProperties wsn:NotificationProducer"
In order to add our five resource properties, we have to add the schema that defines each property and the properties element itself. Listing 4 shows the updates to NewsFeed.wsdl to add the schema for the five article-genre ResourceProperties, as well as the NewsFeedResourceProperties element used to pass them to/from clients.

Listing 4. Updates to NewsFeed.wsdl to add the schema for the five article-genre ResourceProperties

```xml
...<xsd:element name="Weather" type="xsd:string"/>
<xsd:element name="Politics" type="xsd:string"/>
<xsd:element name="Local" type="xsd:string"/>
<xsd:element name="Sports" type="xsd:string"/>
<xsd:element name="Editorial" type="xsd:string"/>
<xsd:element name="NewsFeedResourceProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:Weather" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:Politics" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:Local" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:Sports" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="tns:Editorial" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
...```

Adding new operation providers to the WSDD

The GT4 WS-core is designed in a way such that we can simply plug in existing Java classes that already implement any new functionality required when we extend our portType; this is known as specifying an operation provider. We need to modify the WSDD file to now specify the GT4 providers for the WS-ResourceProperties and WS-Notification portTypes we derived our NewsFeed service portType from. Here is the updated listing for the deploy-server.wsdd source file:

Listing 5. The deploy-server.wsdd file showing updates

```xml
...<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
  xmlns="http://xml.apache.org/axis/wsdd/
  xmlns:java=http://xml.apache.org/axis/wsdd/providers/java
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...```
Adding resource properties/topics to the QNames interface

When we have to refer to any qualified names from either our server or client code, we use the QNames declared in the NewsFeedQNames.java source file. Let’s go ahead and add the qualified names for the genre resource properties (that also serve as topics). Listing 6 shows the updated NewsFeedQNames.java source file.

Listing 6. The updated NewsFeedQNames.java source file

```java
package org.merrill.examples.newsfeedservice.impl;
import javax.xml.namespace.QName;
public interface NewsFeedQNames {
    public static final String NS =
        "http://examples.merrill.org/newsfeed/NewsFeedService";
    public static final QName RESOURCE_PROPERTIES = new QName(NS, "NewsFeedResourceProperties");
    public static final QName RESOURCE_REFERENCE = new QName(NS, "NewsFeedResourceReference");

    /* Insert ResourceProperty Qnames here. */
    public static final QName RP_WEATHER = new QName(NS, "Weather");
    public static final QName RP_POLITICS = new QName(NS, "Politics");
    public static final QName RP_LOCAL = new QName(NS, "Local");
    public static final QName RP_SPORTS = new QName(NS, "Sports");
    public static final QName RP_EDITORIAL = new QName(NS, "Editorial");
}
```

Updating the NewsFeedService implementation
To be able to publish topics, the NewsFeedService class must implement the `org.globus.wsrf.TopicListAccessor` interface. By doing this, the class must now maintain a TopicList used to keep track of all the topics published by our resource and implement a method called `getTopicList()` that exposes it.

**Listing 7. Updates to NewsFeedService.java**

```java
... public class NewsFeedService implements Resource,
    ResourceProperties, TopicListAccessor {
    /* Topic list maintained by this service */
    private TopicList topicList;
    ...
    /**
     * Get the topic list (Required by interface TopicListAccessor)
     * @return The topic list
     */
    public TopicList getTopicList() {
        return topicList;
    }
    ...
}
```

Now we can create the fields of our NewsFeedService class that will serve as our resource properties.

**Listing 8. Updates to NewsFeedService.java showing the declaration of our five article-genre ResourceProperties**

```java
... // Insert resource properties here. */
private ResourceProperty weatherRP;
private ResourceProperty politicsRP;
private ResourceProperty localRP;
private ResourceProperty sportsRP;
private ResourceProperty editorialRP;
...```

Our NewsFeedService constructor needs to accomplish the following things:

- Create the resource property set
- Create the individual resource properties and initialize them
- Initialize the topic list
• Wrap the resource properties in topics and add them to the topic list

The SimpleResourceProperties are wrapped in ResourcePropertyTopics, allowing the same construct to function in both the ResourceProperty list and the TopicList. Listing 9 shows the updated source for the constructor.

Listing 9. The NewsFeedService class constructor in NewsFeedService.java showing the initialization of resource properties and topic list

```java
/* Constructor. Initializes RPs & Topics */
public NewsFeedService() throws RemoteException {  
    /* Create RP set */
    this.propSet = new SimpleResourcePropertySet
                    (NewsFeedQNames.RESOURCE_PROPERTIES);
    
    /* Initialize the RP's */
    try {
        weatherRP = new SimpleResourceProperty
                    (NewsFeedQNames.RP_WEATHER);  
        weatherRP.add("Initial Weather Article");
        politicsRP = new SimpleResourceProperty
                    (NewsFeedQNames.RP_POLITICS);
        politicsRP.add("Initial Politics Article");
        localRP = new SimpleResourceProperty(NewsFeedQNames.RP_LOCAL);
        localRP.add("Initial Local Article");
        sportsRP = new SimpleResourceProperty
                    (NewsFeedQNames.RP_SPORTS);
        sportsRP.add("Initial Sports Article");
        editorialRP = new SimpleResourceProperty
                     (NewsFeedQNames.RP_EDITORIAL);
        editorialRP.add("Initial Editorial Article");
    } catch (Exception e) {  
        throw new RuntimeException(e.getMessage());
    }
    
    /* Create the topic list */
    this.topicList = new SimpleTopicList(this);
    
    /* Wrap the resource properties with ResourcePropertyTopic topics */
    weatherRP = new ResourcePropertyTopic(weatherRP);
    politicsRP = new ResourcePropertyTopic(politicsRP);
    localRP = new ResourcePropertyTopic(localRP);
    sportsRP = new ResourcePropertyTopic(sportsRP);
    editorialRP = new ResourcePropertyTopic(editorialRP);
    
    /* Add the properties/topics to the topic list */
    this.topicList.addTopic((Topic) weatherRP);
    this.topicList.addTopic((Topic) politicsRP);
    this.topicList.addTopic((Topic) localRP);
    this.topicList.addTopic((Topic) sportsRP);
    this.topicList.addTopic((Topic) editorialRP);
    
    /* Add the properties/topics to the property set */
    this.propSet.add(weatherRP);
    this.propSet.add(politicsRP);
```
Although the GT4 SimpleResourceProperty class facilitates an unbounded number of items that can be added to the property (properties can be lists), it would be a mistake to add more than one here because these resource properties are declared to have one (and only one) string.

Section 5. Generating stubs and creating clients

This section covers the process of generating the stub classes that clients will need to interact with the NewsFeed service and the creation of two test clients: an Author to submit new articles to the NewsFeed service and an Editor to be notified with any new articles that match its subscription criteria.

Generating client stubs

Now that we've completed the sources required for the grid service, we can use the GT4IDE to generate the client stubs necessary for us to create our test clients. This is as simple as pressing the Stubs button in the toolbar of the Globus Toolkit IDE perspective. This creates a host of new Java client-side classes, which we can view from the Navigator in the Resources perspective, as shown in Figure 6.

Figure 6. Client-stub files created by GT4IDE's stub-generation feature
This process parses the WSDL and creates proxy classes clients can use to interact with the remote WS-Resource, as well as any datastructures that may need to be passed back and forth (resource properties).

The Editor clients

The Editor.java file is a simple command-line class that subscribes itself to the NewsFeed, requesting that it be notified with any stories that match its topic(s) of interest. When notified, it displays the article to the command console.

The Editor class is composed of three logic components (see Listing 10). The first component, the `main()` method, is responsible for:
• Creating a notification listener object to handle incoming notifications; the
  listener manages its own threads to handle incoming notifications
• Creating an end-point address to the Editor so publishers can deliver
  notifications to it
• Using the subscribe() utility function (described below) to subscribe
  the Editor to the Weather and Local news genres
• Putting its thread of execution into an infinite blocking loop, letting the
  notifications listener handle incoming notifications indefinitely (or at least
  until the user kills the Editor with a Ctrl-C signal)

Listing 10. The main() method in the Editor.java client file

```
... public static void main(String[] args) throws Exception {
    Editor editor = new Editor();

    String serviceURI = args[0];

    // Create a notification listener and instruct it to
    // start listening for publications
    NotificationConsumerManager listener =
        NotificationConsumerManager.getInstance();
    listener.startListening();

    // Register an instance of Editor to handle notifications
    // from the NewsFeed service. It returns an EPR to ourselves
    // so that we can include it in any subscription requests so
    // that the publisher will know how to reach us
    EndpointReferenceType consumerEPR =
        listener.createNotificationConsumer(new Editor());

    // subscribe to WEATHER and LOCAL news
    subscribe(serviceURI, NewsFeedQNames.RP_WEATHER, consumerEPR);
    subscribe(serviceURI, NewsFeedQNames.RP_LOCAL, consumerEPR);

    // Block forever while the listener handles incoming notifications
    while (true) {
        try {
            Thread.sleep(50000);
        } catch (Exception e) {
        }
    }
...```

The second logic component in the Editor class is the subscribe() utility function. Given a NewsFeed portType, a qualified topic (article-genre) name, and the end-point address to the running Editor, it makes a subscription request to the specified NewsFeed service. This is done by creating a subscription request object that contains a TopicExpression indicating what topic the Editor is interested in. Finally, the method invokes the NewsFeed service’s subscribe() operation, as shown in Listing 11.
Listing 11. The subscribe() method in the Editor.java file

```java
/**
 * Subscribe to the given topic on the specified NewsFeed service
 * @param serviceURI the URI denoting the NewsFeed endpoint
 * @param topic QName denoting the genre of articles to subscribe to
 * @param myEpr Our EPR that we want to give to the NewsFeed service
 * to use to identify us
 */
static public void subscribe(String serviceURI, QName topic,
    EndpointReferenceType myEpr) throws Exception {
    // Create a subscription request that indicates we would like
    // to have articles delivered via the Notify operation
    Subscribe request = new Subscribe();
    request.setUseNotify(Boolean.TRUE);
    request.setConsumerReference(myEpr);
    // Create a TopicExpression specifying that we want to
    // subscribe to LOCAL articles
    TopicExpressionType topicExpression = new TopicExpressionType();
    topicExpression.setDialect(WSNConstants.SIMPLE_TOPIC_DIALECT);
    topicExpression.setValue(topic);
    request.setTopicExpression(topicExpression);
    // Create a NotificationProducer portType to communicate
    // with the NewsFeed service
    WSBaseNotificationServiceAddressingLocator notifLocator =
        new WSBaseNotificationServiceAddressingLocator();
    EndpointReferenceType endpoint =
        new EndpointReferenceType(new Address(serviceURI));
    NotificationProducer newsFeedServicePort = notifLocator
        .getNotificationProducerPort(endpoint);
    // Execute the subscribe operation
    newsFeedServicePort.subscribe(request);
}
```

The third component of interest in the Editor class is the deliver() method (see Listing 12). This method serves as a handler function to be called by the NotificationConsumerManager listener whenever a new notification arrives. It simply extracts the notification value and displays it to the console.

Listing 12. The deliver() method in the Editor.java file

```java
/**
 * Receives notification messages
 * @param topicPath The topic path for the topic that generated the
 * notification
 * @param producer The producer endpoint reference
 * the NewsFeed service
 * @param message The notification message
 */
```
public void deliver(List topicPath, EndpointReferenceType producer, Object message) {
    // Cast the message to the appropriate type of element and extract
    // the notification itself
    ResourcePropertyValueChangeNotificationElementType notif_elem =
        (ResourcePropertyValueChangeNotificationElementType) message;
    ResourcePropertyValueChangeNotificationType notif =
        notif_elem.getResourcePropertyValueChangeNotification();
    if (notif != null) {
        System.out.println("An updated article has arrived:");
        System.out.println(notif.getNewValue().get_any()[0].getValue());
        System.out.println();
    }
}

The Author client

The Author.java source file contains the implementation for a simple command-line
class that submits several "articles" to the NewsFeedService. It does this by making
calls to update the "genre" resource property topics. The operation of the Author
client is pretty straightforward; it has three component methods. The first
component, the main() method (see Listing 13), does the following:

• Using the generated locator class, the client creates a proxy stub to the
  remote NewsFeed service.

• Invokes the displayCurrentArticles() method to display the
  NewsFeed service's most recently submitted articles.

• Invokes submitArticle() three times to submit articles to the
  NewsFeed service (a weather, a politics, and a local news story).

• Invokes displayCurrentArticles() again to display the (updated)
  articles maintained by the NewsFeed service.

Listing 13. The main() method in the Author.java client

```java
... public static void main(String[] args) {
    // create a locator that, given an endpoint uri, will
    // facilitate the creation of a proxy stub to that service
    NewsFeedServiceAddressingLocator locator =
        new NewsFeedServiceAddressingLocator();
    try {
        // Create endpoint reference to the NewsFeed service and get
        // the portType proxy to it
        EndpointReferenceType endpoint = new EndpointReferenceType();
```
```
endpoint.setAddress(new Address(args[0]));
NewsFeedPortType newsFeed =
    locator.getNewsFeedPortTypePort(endpoint);

// Perform some operations and print out the resource properties
System.out.println("Getting all stories:");
displayCurrentArticles(newsFeed);

System.out.println("\nUpdating a Local story...");
submitArticle(newsFeed, NewsFeedQNames.RP_LOCAL,
"Local dairy farmer wins blue ribbon for prize" +
"heifer in county fair...");

System.out.println("\nUpdating a Politics story...");
submitArticle(newsFeed, NewsFeedQNames.RP_POLITICS,
"California politician comes in second place in " +
"Strongest Governor contest");

System.out.println("\nUpdating a Weather story...");
submitArticle(newsFeed, NewsFeedQNames.RP_WEATHER,
"Tornado destroys local dairy farm, prized " +
"heifer missing...");

System.out.println("\nGetting all stories:");
displayCurrentArticles(newsFeed);
}
```

The second section of the Author client is the `displayCurrentArticles()` utility method. It creates a request object and inserts into it the qualified names of the `ResourceProperties` it's interested in. Then it invokes `getMultipleResourceProperties()` on the remote NewsFeed service and iteratively displays the values of the returned `ResourceProperties`.

**Listing 14. The `displayCurrentArticles()` method in the Author.java client**

```
.../**
* This method retrieves all of the current articles maintained
* by the NewsFeed service.
*/
static private void displayCurrentArticles
    (NewsFeedPortType newsFeed)
    throws Exception {
    GetMultipleResourceProperties_Element request;
    GetMultipleResourcePropertiesResponse response;

    // create a request object with the names of all of the
    // resource properties that we're interested in
    QName[] resourceProperties = new QName[] {
        NewsFeedQNames.RP_WEATHER,
        NewsFeedQNames.RP_SPORTS,
        NewsFeedQNames.RP_LOCAL,
```
NewsFeedQNames.RP_POLITICS,
NewsFeedQNames.RP_EDITORIAL
};
request = new
GetMultipleResourceProperties_Element(resourceProperties);
// Call the GetMultipleResourceProperties operation on the NewsFeed
response = newsFeed.getMultipleResourceProperties(request);
// Display the returned resource properties
for (int i = 0; i<response.get_any().length; i++)
{
    String genre = response.get_any()[i].getLocalName();
    String text = response.get_any()[i].getValue();
    System.out.println(genre +": " + text);
}
...

The third component of the Author client is the submitArticle() utility method. It creates an update request with a single update object and inserts into it the new value and the qualified name for the article-genre ResourceProperties it's about to update. Then it invokes setResourceProperties() on the remote NewsFeed service.

Listing 15. The submitArticle() method in the Author.java client

/**
 * This method submits the specified article text to the given
 * NewsFeed service that is related to the provided topic. The
 * topic is expected to correlate to a resource property, whose
 * value will be given that of the new article text
 * @param newsFeed NewsFeedPortType proxy stub with which to
 * manipulate the service with
 * @param topicGenre QName identifying the genre of the specified article
 * @param articleText String containing the text of the article to submit
 **/
static private void submitArticle(NewsFeedPortType newsFeed,
    QName topicGenre, String articleText) throws Exception {
    // Prepare an update object with a single update request
    UpdateType update = new UpdateType();
    MessageElement msg = new MessageElement(topicGenre, articleText);
    update.set_any(new MessageElement[] { msg });
    SetResourceProperties_Element request =
        new SetResourceProperties_Element();
    request.setUpdate(update);
    // call the SetResourceProperties operation
    newsFeed.setResourceProperties(request);
}

Creating the GAR for deployment

Before we can deploy our service in a hosting container (in Tomcat or the GT stand-alone container), we have to package everything up in a GAR (the grid archive
that contains all the files and information the Web services container needs to deploy our service). To do this, we can simply use the GAR button in the toolbar of the Globus Toolkit IDE perspective.

NOTE: I had to do two things for the .GAR file to be properly generated by GT4IDE:

- Alter the jars target in the build.xml buildfile to depend on jarStubs, jar, dist, instead of just jarStubs, jar
- Create a \Dev\Tutorial\eclipse\workspace\NewsFeedService\NewsFeedService\etc directory in my Eclipse workspace

Section 6. Testing the NewsFeed service

This section illustrates the operation of the Author and Editor clients to test the NewsFeed service.

Client execution

Once we have the EAR deployed and running inside a suitable container (Tomcat or the stand-alone GT4 container), we can test our service by setting up an Editor that wishes to be notified whenever a local or weather story is submitted and an Author that submits several stories (local, political, and weather stories, to be specific).

Listing 16. Output from the Author client

D:\>\j2sdk1.4.2_08\bin\java -classpath D:\Dev\Tutorial\eclipse\workspace\NewsFeedService \NewsFeedService\build\classes;%CLASSPATH% org.merrill.examples.newsfeedservice.impl.Author http://192.168.0.107:8080/wsrf/services/examples/NewsFeedService

Getting all stories:
Weather: Initial Weather Article
Sports: Initial Sports Article
Local: Initial Local Article
Politics: Initial Politics Article
Editorial: Initial Editorial Article

Updating a Local story...
Updating a Politics story...
As predicted, the first reading of articles from the resource properties of the NewsFeed service returns their initial settings. Then, after submitting three stories (a local, a political, and a weather article), the output shows that the resource properties have correctly been updated.

A few milliseconds after submitting, the Editor client program displays what appears in Listing 17.

**Listing 17. Output from the Editor client**

```
D:\Dev\Tutorial\GTK>\j2sdk1.4.2_08\bin\java -classpath D:\Dev\Tutorial\eclipse\workspace\NewsFeedService \NewsFeedService\build\classes;%CLASSPATH% org.merrill.examples.newsfeedservice.impl.Editor http://192.168.0.107:8080/wsrf/services/examples/NewsFeedService

An updated article has arrived:
Local dairy farmer wins blue ribbon for prize heifer in county fair...

An updated article has arrived:
Tornado destroys local dairy farm, prized heifer missing...
```

Our editor has correctly been notified of the updates to the two topics that it subscribed to: the Local and the Weather and the topics (but not the Politics topic).

---

**Section 7. Summary**

This tutorial demonstrated how to create a simple GT4 grid service that leverages WS-Notification to implement a publish-subscribe pattern. We gave a step-by-step walk-through of how to create, implement, and use the NewsFeed Java project, focusing on:
• A background of the WSRF and WSN families of specifications and the Globus Toolkit

• Setting up Eclipse, GT4, Tomcat, GT4IDE, and the other necessary plug-ins and tools

• Creating and completing the project's source files, illustrating the changes necessary to introduce WSN support to a simple grid service

• The simplicity of using the combined ResourceProperty/Topic construct and leveraging the GT4 WS-ResourceProperty and WSN provider implementations

• Using the GT4IDE to orchestrate the automatic generation of the remaining artifacts, assembling the grid archive (GAR), and deploying the grid service into the Web services container

• Running and testing the grid service

The publish-subscribe pattern is a powerful and frequently used approach for architecting loosely coupled services and systems. The WSN group of specifications does a good job of enabling this pattern in the realm of Web services development, which is significant because Web services was not originally intended for event-driven programming. In this tutorial, we've only covered a small amount of what WSN has to offer, and we hope it has provided a simple blueprint from which to base more complicated publish-subscribe services on.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code for the NewsFeed application</td>
<td>gr-pubsubNFSSources.zip</td>
<td>22KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

*Information about download methods*
Resources

Learn

- Read the **WS-Notification specification working drafts**, maintained by OASIS.
- "**Publish Subscribe Notification for Web Services**" discusses notification patterns and the goals for WSN.
- See the developerWorks tutorial, "**Understanding WSRF, Part 3**" for publish-subscribe with WS-Notification information.
- Learn more about the **WSRF specification working drafts**, maintained by OASIS.
- See the four-part series on WSRF from developerWorks titled "**Understanding WSRF**."
- Read The Globus Alliance's **overview of WSRF**.
- Learn more about installing the Globus Toolkit by reading the **GT4 Admin Guide**. You should also check out the rest of the **Globus Toolkit documentation**.
- Read the **Globus Toolkit 4 Programmer's Tutorial**.

Get products and technologies

- Download Globus Toolkit V4.0.1 from **Globus.org**.
- The **GT4 IDE** is available.
- Get the **Eclipse Platform** from Eclipse.org.
- Get the **Apache Tomcat server** and the **Sysdeo Eclipse Tomcat Launcher Plug-in**.

About the author

Duane Merrill

Duane Merrill has been developing grid computing and distributed data integration platforms for more than five years. He has been a contributor to the Legion Project at the University of Virginia and a core developer for the Avaki Corp.'s distributed enterprise information integration (EIi) product Avaki. He is currently working toward his doctorate in computer science at the University of Virginia.