S-QoS4WS: Secure Quality of Service for Web Services Using the 3rd Party Certifications

Phil Bonderud[1], Sam Chung[1], Barbara Endicott-Popowski[2], Don McLane[1]

[1]Computing & Software Systems
Institute of Technology
University of Washington, Tacoma
{bonderud, chungsa, dmclane @ u.washington.edu}

[2]Center for Information Assurance and Cybersecurity (CIAC)
University of Washington, Seattle
endicott@u.washington.edu

Abstract

The purpose of this research is to propose a mechanism that enables trustworthy relationships among service producers and consumers. Service producers advertise their services through the Universal Description, Discovery, and Integration (UDDI). The reliability of information contained within the UDDI cannot be trusted without being able to validate the authenticity of claims made by both service consumers and producers. From a Quality of Service (QoS) perspective, a consumer of a service needs to know if a service will meet his or her requirements for which various web services are contracted. In terms of a security perspective, a service consumer that obtains services for which they are unable to pay establishes a security concern for service producers. The requirement of the secure Quality of Services for Web Services (S-QoS4WS) needs a mechanism to enable service consumers and service producers to establish trust in their prospective partners prior to engaging in any service for which payment or quality of service is required. For this purpose, we propose a mechanism that utilizes 'PublisherAssertion' tags within the UDDI to satisfy Security and QoS issues, which are considered out-of-scope for the latest version 3 UDDI release. By using the existing architecture for version 3 of the UDDI, it is possible for service consumers and producers to establish a trustworthy relationship, with respect to their own interests, prior to engaging service transactions. The benefits of using this approach include the reuse of existing UDDI mechanism, the establishment of trust, and a solution for non-repudiation.

1. Introduction

There are two issues, currently considered out of scope for the latest version of the *Universal Description, Discovery, and Integration (UDDI* v3), which motivated this research: establishing trust and non-repudiation. Within web services, an argument for Quality of Service (QoS) holds that a third party mechanism is required to make clear to service consumers the quality of service a service producer is offering [2]. It is unrealistic to expect that a business will search for web services based on the expected functional requirements alone, listed in a UDDI registry, and invoke a service, without knowing that its QoS requirements will be met [2]. The purpose of this project is to show that: 1) It is possible to address security and QoS issues within the current UDDI framework; 2) An alternative application of existing UDDI mechanisms can support service consumer validations and service producer certifications; And, 3) this approach addresses current unresolved UDDI issues regarding trust and non-repudiation.

Since an objective of this project is to avoid requesting changes to the UDDI architecture, the use of this approach should be considered optional as with most all things related to the UDDI By using this approach the solution for the issue trust and non-repudiation is solvable today without any involvement from the consortium of organizations which guide the development of a UDDI specification

2. Previous Work

In the paper "A Model for Web services Discovery with QoS," Shuping Ran [2] explains how the rise in use of web services has been slow because of QoS issues relating to the UDDI and the services listed within it. Ran's key arguments revolve around questions asked by a typical service consumer, such as "how will I know that the web service will meet my performance requirements such as 2ms response time? Will the web service be reliable for my mission-critical system's implementation?" Until these questions are addressed, it is unrealistic to expect that a business will want to search for web services based on the expected functional requirements in a UDDI registry and invoke that service without the assurance of knowing the expected quality of service will be met before hand.

The goal of Shuping Ran's paper was to propose a new service discovery model where validation of a service can be used as a constraint when searching for web services, as a means of instilling confidence in clients who use the service. Ran's proposition was to use a third party certifier to validate QoS claims made by the service producer and to either substantiate or downgrade UDDI advertised claims. The burden for validation is placed on the UDDI registrar and a requirement to check for certification statements was specified as a prerequisite for publishing advertisements in the UDDI.

In the paper "UDDI and WSDL Extensions for Web Service: A Security Framework," Carlisle Adams and Sharon Boeyen [3] extend the same argument that web services within the UDDI need to be trusted. Adams and Boeyen argued that facilitating the use of web services on a large scale requires that security must be added to insure registry security within the UDDI that transactions be conducted in a trustworthy manner and that infrastructure linkages ensure that services are implemented in a trustworthy manner. A main point by the authors is that if information within the UDDI cannot be trusted then nobody will use it. One aspect of Adams and Boeyen's argument was implemented in the latest Version 3 UDDI update: digital signatures [8]. Through the use of digital signatures users of the UDDI can be guaranteed that UDDI entries have not been tampered with nor corrupted.

Among the security enhancements proposed by Adams and Boeyen was a brief argument for an authentication infrastructure which used public and private keys to establish identities among consumers and producers. One of the purposes of this authentication mechanism was to offer confidence to both service producers and consumers that the business entities are who they appear.

In the paper "A Snapshot of Public Web services," Fan Jianchun and Kambhampati Subbarao [5] discuss the growth of web services and acknowledge that there are numerous propositions offered as to which direction web services should grow. Jianchun and Kambhampati explain that each position extends the perspective of the author from which it was proposed. Some believe that web services will remain in the public domain and some see it as only within private domains. In regards to the UDDI, the authors explain that the registry provides facilities for service producers to advertise their services in some standard industry taxonomy and also for the user to query the desired service profile. They state that an issue with UDDI registries is that the concept has not yet matured. Input to such registries depends on user/developer input and anyone can enter anything and make any type of claim about their service.

3. UDDI Background

The Organization for the Advancement of Structured Information Standards (OASIS) establishes standards for the UDDI [7, 8, 9, 10, 11, 12]. The UDDI protocol is an eXtensible Markup Language (XML) based registry for locating web services. Interactions with the UDDI are conducted through the transmission of XML based Simple Object Access Protocol (SOAP) messages. The goal of the UDDI is to provide an automated way for service consumers to discover and consume services; ultimately without any manual intervention. Today, when a service consumer locates a service within the UDDI, the consumer must manually inspect each service's specification, listed in the service's Technical Model (tModel). If satisfied with these specifications, the service consumer locates the

Web services Description Language (WSDL) file, connects to the service, and tests that the service's performance matches its stated specifications. The WSDL is an XML file that describes the interface through which service consumers will access the service. The WSDL may also contain Uniform Resource Identifiers (URIs) that identify additional sources of information about the service and/or service producer [11].

In the beginning, attention was focused on the concept of a "Universal Business Registry" (UBR) that represented "a master directory of publicly available e-commerce services" [8]. A common analogy for this approach was to compare the UBR to a set of phone books where white-pages represented company contact information; yellow-pages categorized businesses by services; and green-pages contained technical information about exposed services [8]. Today the acronym UBR stands for UDDI Business Registry. There are several different types of UDDIs such as public, private, and shared/semi-private UDDIS.

The UDDI protocol is a key element among a group of standards that comprise web services. Version 3 of the UDDI specification is the latest version released. Version 3 defines a set of standards for publishing and discovering network-based software components within the Service-Oriented Architecture (SOA) [9]. The UDDI uses XML because it is platform neutral and it allows hierarchical relationships to be described in a natural way. "The UDDI business registry is intended to serve as a global all-inclusive listing of businesses and their services" [8]. The ultimate goal of the UDDI is to provide a mechanism that will enable automated search, discovery, and consumption of services.

Numerous UDDI registries exist. The contents of each record may be protected through the use of digital signatures (DSIG) that provide a mechanism to establish whether an entry within a UDDI has been altered (corrupted data or tampering) since it was published. DSIGs also enable proof of ownership for a registry entry. The DSIG specification for this signature was created by World Wide Web Consortium (W3C) [8] and like all things UDDI related, using DSIGs is optional.

4. Service Producer and Service Certification

The burden of certifying the quality of a service offered by a producer is currently the responsibility at the discretion of the consumer. Automated search, discovery, verifying specifications, and service consumption remains beyond the reach of current technology [6]. Researchers from a QoS background proposed modifying the UDDI to enable service producers the ability to substantiate quality related claims about their services, by including a certification claim from a third party validation service [2]. This third party entity examines both the QoS claim from the service producer and tests the validity of these claims against the actual service [2]. At the conclusion of this certification process, the certifying entity issues a certificate to the service producer that either substantiates the service producer's QoS claim about their service or downgrades it, which is shown in Figure 1. With respect to this research, the certificate generated by the third party certifier is in the form of a simple web service which announces the results of the certification.

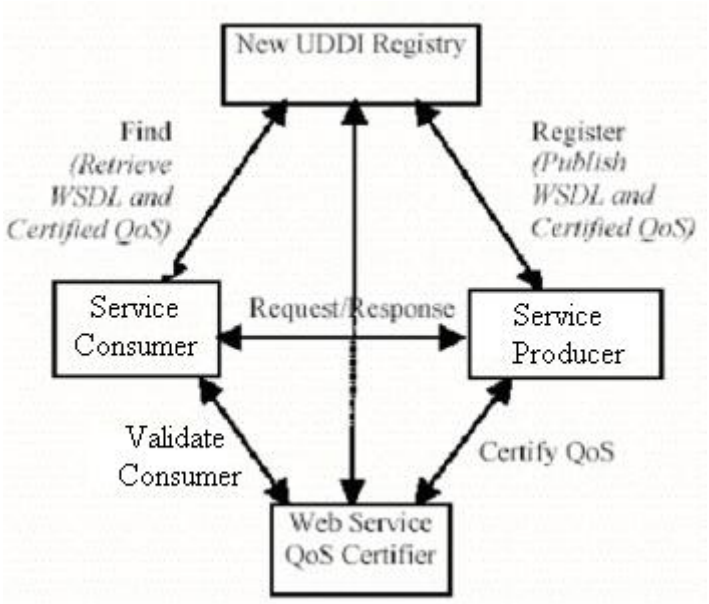


Figure 1: Working within the existing UDDI (v3) framework, service consumers are capable of validating QoS claims made by service producers through the use of a third party certification entity. This figure represents "a new web services registration and discovery model" [2].

Unlike the proposed mechanisms set forth by Shuping Ran[2], this research takes the position that the burden of proof that services are certified is not the responsibility of the UDDI operator. Here, certification of services are optional and left to the discretion of each service producer.

5. Service Consumer and Consumer Validation

UDDI research does not examine the role of the service consumer with respect to enabling any sort of consumer validation. The UDDI's goal is to allow an automated mechanism for service consumers to discover and connect to services offered by service producers. No specifications exist which identify a uniform way for service producers to validate consumers, so that producers can hold a degree of assurance that they will get paid for services consumed. Validation of the service consumer is a component of this research.

This research mirrors the general approach used by service producers and service certification entities to offer consumer validations through the use of consumer validation entities.

6. The Secure Quality of Service for Web Services (S-QoS4WS) Approach

S-QoS4WS takes into consideration security and QoS issues with respect to establishing trust and non-repudiation. The approach adds an optional third party entity to the web services paradigm whose sole purpose is to certify information about each respective business partner. The third party service certifier certifies that services offered by a service producer meet the specifications used to describe the service in the UDDI. The third party consumer validation entity authenticates that its service consumer partner is a trustworthy and legitimate business. Each third party entity is expected to publish its own web service whose sole purpose is to provide an automated way of obtaining information.

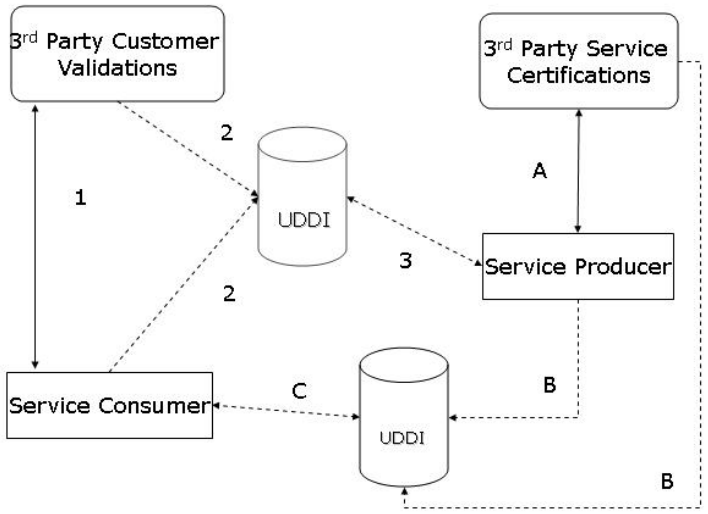


Figure 2: A general overview of the S-QoS4WS approach. Two UDDI are shown for diagram clarity.

In Figure 2, solid lines represent interactions that require human intervention. It is expected that in order for a service to be adequately certified or a consumer to be validated, that some degree of human involvement will be required. Dashed lines represent transactions that are fully automated.

S-QoS requires that a service producer select a third party entity (A) which will certify that any statistics and requirements it wishes to advertise in the UDDI, about a service, are accurate. This communication is expected to require human involvement, which is indicated by the solid line in Figure 2.

Upon reaching final agreement concerning a service's certification (B), the certifying entity publishes a web service to a UDDI. This service, published by the certifier, holds the results of a service's certification. The service producer (B) also publishes its service to the UDDI, if it has not already done so. Both the service producer and the third party certification entity make identical 'publisherAssertions' for this service, which will be explained in detal in the next section. By making identical 'publisherAssertions' for this service (C), service consumers can query the UDDI for 'status:complete' certified services.

S-QoS mirrors the interactions between the service producer and its certification entity to produce consumer validations (1 – 3). Whether or not a unique UDDI is used as diagramed, which caters only to service consumers, is irrelevant to this research and not a requirement for the success of this approach. Equivalent to communications represented by line A, communications between a service consumer (1) and its respective third party service consumer validation entity is expected to require human involvement.

Upon reaching final agreement over the information to be published (2), the validation entity publishes a web service to a UDDI that holds the results of a consumer's validation. The service consumer (2) also publishes an informational service to the UDDI that represents itself, if it has not already done so. Both the service consumer and the third party validation entity make identical 'publisherAssertions' for the consumer. By making identical publisherAssertions for the consumer (3), service producers obtain an added bonus of being able to query a UDDI for 'status:complete' validated consumers. This added bonus enables service producers to proactively market their services to viable organizations.

Within each UDDI businesses have the option of providing a service description statement. Service consumers will enter 'Service Consumer' as their descriptor, consumer evaluators will enter 'Consumer Validation', service certification entities will use 'Service Certification', and service producers default to any description.

7. Implementing S-QoS4WS

7.1 Using the 'PublisherAssertion'

The 'PublisherAssertion' is a UDDI entity type which describes the relationship between two business entities. The defined purpose for this type of entity is that there are "many instances where multiple divisions within a large organization or a group of organizations want to make the relationship between them known in order to facilitate discovery of the services they provide. Individual divisions or organizations are represented in the UDDI by the businessEntity tag. The entity type 'publisherAssertion' describes the relationship between two business entities. It is important to note that two organizations must assert the exact same relationship through the 'publisherAssertion' for that relationship to be publicly available using 'status:complete.' This disallows the situation where one organization claims a relationship with another where in fact there is none" [6]. Business entities can publish multiple advertisements within the UDDI that enables a many to many relationship among consumers, producers and their respective third party entities.

UDDI specifications specifically state that the inclusion of 'publisherAssertions' is not intended to establish trust. We disagree. An argument we hold is that by using 'publisherAssertions' in the manner described by this research the publisherAssertions can establish trust. When used among service consumers and a third party consumer validation entity, the publisherAssertions also solve the issue of non-repudiation.

The contents of two publisherAssertions, without SOAP message formatting, are given in Figure 3. Figure 3 shows two partial 'publisherAssertion' statements without SOAP message tags, namespace URIs, or authorization tokens. Notice that with the exception of each publisherAssertion descriptive reference, both publisherAssertion statements are identical. Each party represented by the publisherAssertion is identified by the ToKey or FromKey. These business keys are unique for each business within the UDDI.

Publisher Assertion From Service Certifier:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Publisher Assertion From Service Producer:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Figure 3: Contents of a set of valid UDDI publisherAssertions between a third party certification entity and a service producer, with SOAP message tags removed.

In order for these two business entities to make this assertion, an automated request is made to the UDDI to obtain an authorization token. The authorization token is a key which has a short lifespan and has the format: authToken:9882AFC0-C626-11DB-B794-D5BF3565FF5A. The UDDI uses authorization tokens for every transaction that attempts to modify a business entity record. Each authorization token uniquely identifies a specific business entity as well.

When used in conjunction to publisherAssertions the UDDI validates that the authorization token matches the business key for the business entity attempting to create a publisherAssertion. If there is a mismatch or the authorization token's lifespan has expired, the transaction is rejected by the UDDI.

7.2. Building a UDDI Advertisement

This research makes use of open source technologies in order to show a complete set of SOAP message transactions necessary to generate a basic UDDI service advertisement and publisher assertion. For the most part, these technologies lacked sufficient instructions in order to readily use them for this project. As a result, tutorials were created and submitted to the respective organizations in order to assist future developers with their use.

The development environment for this research includes:

- Web service engine - Apache-Tomcat 5.5.20
-
- Integrated Development Environment (IDE) for developing web services - Eclipse Web Tools Platform (WTP) 1.5
- UDDI server for managing web services publication and discovery - jUDDI 0.9rc4
- Database Management Server (DBMS) for jUDDI data - MySQL 5.0.26 (win32)
-
- Java Development Kit (JDK) - JRE & JDK 1.5.0_9

Within jUDDI, authorization tokens (keys) are required which identify each respective producer within a UDDI. In order to obtain authorization tokens a new producer must register with the UDDI. Below are two SOAP messages that perform this task. The first illustrates the message to jUDDI, creating a new publisher and the second message shows a successful response from jUDDI.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_publisher generic="1.0" xmlns="urn:juddi-org:api_v1">
      <authInfo>authToken:D655CFD0-C621-11DB-B794-ACAB1428A8CF</authInfo>
      <publisher
        publisherID="Service Producer"
        publisherName="Service Producer"
        admin="true"
        enabled="true"
        emailAddress="Service Producer@pbonderud.com"/>
    </save_publisher>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <publisherDetail generic="1.0" operator="jUDDI.org" xmlns="urn:juddi-org:api_v1">
   <publisher admin="true" emailAddress="Service Producer@pbonderud.com" enabled="true" publisherID="Service Producer"
publisherName="Service Producer"/>
  </publisherDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Once created the publisherID for the custID in the following SOAP message transaction, the transaction generates an authorization token which uniquely represents this particular publisher. A new authorization token is required for every transaction within JUDDI. The authorization token also times out if not used quickly.In the following SOAP transaction the publisher (here representing a service producer) is prompted for a credential, such as a password. The password feature is not operational within JUDDI and its use causes the transaction to fail every time.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_authToken generic="2.0" xmlns="urn:uddi-org:api_v2"
      userID="Service Producer"
      cred="***"/>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <authToken generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <authInfo>authToken:2D98D4E0-C622-11DB-B794-A48302DAB07D</authInfo>
  </authToken>
 </soapenv:Body>
</soapenv:Envelope>
```

Next we need to obtain a unique business key and in order to do that we must register a business entity with jUDDI. Registering for a business service takes just a few basic contact details such as a business name, description (which is where standard terms could be used to identify service consumers), contact useType (the type of key being obtained), contact person, telephone number, and email address.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_business generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:783759D0-C623-11DB-B794-91C93F1E270C</authInfo>
      <businessEntity businessKey="">
        <name>Service Producer</name>
        <description>Service Producer Services</description>
        <contacts>
          <contact useType="businessEntity">
            <personName>Mr. Service Producer</personName>
            <phone>360-895-2199</phone>
            <email>Service Producer@pbonderud.com</email>
          </contact>
        </contacts>
      </businessEntity>
    </save_business>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <businessDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <businessEntity authorizedName="Service Producer" businessKey="80604770-C623-11DB-B794-D20C319D25C5"
operator="jUDDI.org">
    <discoveryURLs>
     <discoveryURL useType="businessEntity">http://localhost:8080/juddi/uddiget.jsp?businesskey=80604770-C623-11DB-B794-
D20C319D25C5</discoveryURL>
    </discoveryURLs>
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <contacts>
     <contact useType="businessEntity">
      <personName>Mr. Service Producer</personName>
      <phone>360-895-2199</phone>
      <email>Service Producer@pbonderud.com</email>
     </contact>
    </contacts>
   </businessEntity>
  </businessDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

In the next example, a unique Service Key is obtained through the registration of a service with jUDDI. As with each step a new authorization token is generated using the producer's initial registration information. In this successive step the Business Entity key, which was generated in the previous step, is used to assist with tying the two sets of registrant information together.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <save_service generic="2.0" xmlns="urn:uddie-org:api_v2">
   <authInfo>authToken:7ACBA3D0-C624-11DB-B794-9786F7ABC0B2</authInfo>
   <businessService businessKey="80604770-C623-11DB-B794-D20C319D25C5" serviceKey="">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <bindingTemplates>
     <bindingTemplate bindingKey="">
      <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="***">
        <instanceDetails>
         <overviewDoc>
          <overviewURL>***</overviewURL>
         </overviewDoc>
        </instanceDetails>
       </tModelInstanceInfo>
      </tModelInstanceDetails>
     </bindingTemplate>
    </bindingTemplates>
   </businessService>
  </save_service>
 </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <serviceDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <businessService businessKey="80604770-C623-11DB-B794-D20C319D25C5" serviceKey="81E48100-C624-11DB-B794-
B50CD54D7639">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <bindingTemplates>
     <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-
B50CD54D7639">
      <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="***">
        <instanceDetails>
         <overviewDoc>
          <overviewURL>***</overviewURL>
         </overviewDoc>
        </instanceDetails>
       </tModelInstanceInfo>
      </tModelInstanceDetails>
     </bindingTemplate>
    </bindingTemplates>
   </businessService>
  </serviceDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Continuing with the successful registration of our service we must obtain a binding key and a new service key, which assist with the registration of the actual service. In order to obtain this binding key we first needed to register the producer, obtain a new authorization token, and provide the producer's business and service keys.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_binding generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:9F85FE90-C625-11DB-B794-FEFC5EE8D774</authInfo>
      <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-B50CD54D7639">
        <description xml:lang="en">Service Producer Services</description>
        <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      </bindingTemplate>
    </save_binding>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <bindingDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-B50CD54D7639">
    <description xml:lang="en">Service Producer Services</description>
    <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
   </bindingTemplate>
  </bindingDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Next we obtain a tModel key by registering the actual service. When registering the actual service, the authorization key we generate at each step identifies this service to its actual service producer.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
   <authInfo>authToken:9882AFC0-C626-11DB-B794-D5BF3565FF5A</authInfo>
   <tModel tModelKey="">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <overviewDoc>
     <description>***</description>     <overviewURL>platform:/resource/Servers/MyWebClass.wsdl</overviewURL>
    </overviewDoc>
    <identifierBag>
     <keyedReference tModelKey="***" keyName="***" keyValue="***" />
    </identifierBag>
    <categoryBag>
     <keyedReference tModelKey="***" keyName="***" keyValue="***" />
    </categoryBag>
   </tModel>
  </save_tModel>
 </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <tModelDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <tModel authorizedName="Service Producer" operator="jUDDI.org" tModelKey="uuid:A78D6500-C626-11DB-B794-
96E880134AAD">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <overviewDoc>
     <description>***</description>
<overviewURL>platform:/resource/Servers/MyWebClass.wsdl</overviewURL>
    </overviewDoc>
    <identifierBag>
     <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
    </identifierBag>
    <categoryBag>
     <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
    </categoryBag>
   </tModel>
  </tModelDetail>
 </soapenv:Body>
```

In addition to these steps listed other services were also registered so that more than one service existed within jUDDI. These other services are not shown here because their introduction would provide too much redundancy. The other service, not shown, is called *Service Certifier*.

Creating the publisherAssertion requires two publishers who make identical statements about each other. These identical statements may be within the same UDDI or across two different UDDIs. The format for this publisherAssertion is as follows:

Publisher Assertion From Service Certifier:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Publisher Assertion From Service Producer:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Notice that the two publisherAssertions are truly identical. The mechanism that prevents bogus claims is that each publisher assertion also requires an authorization token/key. The authorization token authenticates the Business Key. If a business key is used which does not authenticate with the authentication token a SOAP fault error is generated and the transaction is refused by jUDDI.

Following with the set of SOAP message examples below is the publisherAssertion transaction for service producer.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <add_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
      <authInfo>authToken:0A001290-C628-11DB-B794-944076723DD3</authInfo>
      <publisherAssertion>
        <fromKey>5FB39040-C623-11DB-B794-FB86F3278740</fromKey>
        <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
        <keyedReference tModelKey="uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF" keyName="Service Certifier" keyValue="parent-
child" />
      </publisherAssertion>
    </add_publisherAssertions>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
```

Finally, prospective service consumers who wish to identify complete publisherAssertions can do so by performing the following transaction.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_assertionStatusReport generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:643D2D50-CAD4-11DB-AD50-9C9D409BE183</authInfo>
      <completionStatus>status:complete</completionStatus>
    </get_assertionStatusReport>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <assertionStatusReport generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <assertionStatusItem completionStatus="status:complete">
    <fromKey>5FB39040-C623-11DB-B794-FB86F3278740</fromKey>
    <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
    <keyedReference keyName="Service Certifier" keyValue="parent-child" tModelKey="uuid:3AE5D7C0-C626-11DB-B794-
F8CC410B3ADF"/>
    <keysOwned>
     <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
    </keysOwned>
   </assertionStatusItem>
  </assertionStatusReport>
 </soapenv:Body>
</soapenv:Envelope>
```

8. Conclusion

The S-QoS4WS approach makes use of existing mechanisms within UDDI version 3 to resolve current issues involving trust and non-repudiation. By using publisherAssertions, we introduce third party customer validations and service certifications. Consumers of web services can trust that the services discovered within a UDDI will meet their requirements. Furthermore, service producers can verify that consumers interested in consuming their services are authentic and reputable business entities. An added bonus to the approach described by this research is that service producers can also seek out potential customers for their services rather than simply waiting to be discovered.

Also, the S-QoS4WS approach opens the doorway to new startup companies and existing enterprises who wish to get involved with web services by providing these third party services. Together, the S-QoS4WS provides an optional tool to move web services toward a fully automated environment. The S-QoS4WS uses existing mechanisms currently available within the UDDI and applies them in a slightly different way, counter to their intended purpose, in order to satisfy existing UDDI issues regarding non-repudiation and establishing trust.

References

[1] Rompothong, Pornpong, and Senivongse, Twittie. (2003) A Query Federation of UDDI Registries. Book Title: ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies. pp 561-566

[2] Ran, Shuping. (2003) A Model for Web Services Discovery with Qos.
ACM Press. Volume 4. No. 1. pp 1-10.

[3] Adams, Carlisle. and Boeyen, Sharon. (2002) UDDI and WSDL Extensions for Web Service: A Security Framework. ACM Press. Book. ISBN: 1-58113-632-3. pp 30-35

[4] Banerjee, Sujata. and Basu, Sujoy. et. al. (2005) "Scalable Grid Service Discovery Based on UDDI. Book. ISBN 1-59593-269-0. pp 1-6.

[5] Jianchun Fan; Subbarao Kambhampati. (2005). A Snapshot of Public Web Services. ACM Press. Vol. 34, No. 1. pp. 24- 32.

[6] Chatterjee, Sandeep Ph.D. and Webber, James Ph.D. "Developing Enterprise Web Services An Architect's Guide." Pearson Education, Prentice Hall Professional Technical Reference, Upper Saddle River, NJ. 2004. pp 1 – 143.

[7] OASIS UDDI, http://www.uddi.org/faqs.html#whatis

[8] The Stencil Group. (2002) "The Evolution of UDDI" UDDI.org White Paper.
URL: http://www.uddi.org/whitepapers.html

[9] OASIS (2004) "Introduction to UDDI: Important Features and Functional Concepts"
White Paper. URL: URL: http://www.uddi.org/whitepapers.html

[10] OASIS (2006) "Web Services Security 1.1"
Specification Standards, URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss#technical

[11] W3C (2006) "UDDI in one slide"
www.w3.org/2003/Talks/0818-msm-ws/slide23-0.html

[12] OASIS (2006) "UDDI Version 3 Feature List"
www.uddi.org/pubs/uddi_v3_features.htm#_Toc10457167

[13] Alan "Re: How to Publish to My Own jUDDI Server?" http://dev.eclipse.org/newslists/
news.eclipse.webtools/msg11863.html

Appendix A – Building a UDDI Advertisement within JUDDI

Within jUDDI, authorization tokens (keys) are required which identify each respective producer within a UDDI. In order to obtain authorization tokens a new producer must register with the UDDI. Below are two SOAP messages that perform this task. The first illustrates the message to jUDDI, creating a new publisher and the second message shows a successful response from jUDDI.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_publisher generic="1.0" xmlns="urn:juddi-org:api_v1">
      <authInfo>authToken:D655CFD0-C621-11DB-B794-ACAB1428A8CF</authInfo>
      <publisher
        publisherID="Service Producer"
        publisherName="Service Producer"
        admin="true"
        enabled="true"
        emailAddress="Service Producer@pbonderud.com"/>
    </save_publisher>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <publisherDetail generic="1.0" operator="jUDDI.org" xmlns="urn:juddi-org:api_v1">
   <publisher admin="true" emailAddress="Service Producer@pbonderud.com" enabled="true" publisherID="Service Producer"
publisherName="Service Producer"/>
  </publisherDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Once created the publisherID for the custID in the following SOAP message transaction, the transaction generates an authorization token which uniquely represents this particular publisher. A new authorization token is required for every transaction within JUDDI. The authorization token also times out if not used quickly.In the following SOAP transaction the publisher (here representing a service producer) is prompted for a credential, such as a password. The password feature is not operational within JUDDI and its use causes the transaction to fail every time.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_authToken generic="2.0" xmlns="urn:uddi-org:api_v2"
      userID="Service Producer"
      cred="***"/>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <authToken generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <authInfo>authToken:2D98D4E0-C622-11DB-B794-A48302DAB07D</authInfo>
  </authToken>
 </soapenv:Body>
</soapenv:Envelope>
```

Next we need to obtain a unique business key and in order to do that we must register a business entity with jUDDI. Registering for a business service takes just a few basic contact details such as a business name, description (which is where standard terms could be used to identify service consumers), contact useType (the type of key being obtained), contact person, telephone number, and email address.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <save_business generic="2.0" xmlns="urn:uddi-org:api_v2">
   <authInfo>authToken:783759D0-C623-11DB-B794-91C93F1E270C</authInfo>
   <businessEntity businessKey="">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <contacts>
     <contact useType="businessEntity">
      <personName>Mr. Service Producer</personName>
      <phone>360-895-2199</phone>
      <email>Service Producer@pbonderud.com</email>
     </contact>
    </contacts>
   </businessEntity>
  </save_business>
 </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <businessDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <businessEntity authorizedName="Service Producer" businessKey="80604770-C623-11DB-B794-D20C319D25C5"
operator="jUDDI.org">
    <discoveryURLs>
     <discoveryURL useType="businessEntity">http://localhost:8080/juddi/uddiget.jsp?businesskey=80604770-C623-11DB-B794-
D20C319D25C5</discoveryURL>
    </discoveryURLs>
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <contacts>
     <contact useType="businessEntity">
      <personName>Mr. Service Producer</personName>
      <phone>360-895-2199</phone>
      <email>Service Producer@pbonderud.com</email>
     </contact>
    </contacts>
   </businessEntity>
  </businessDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

In the next example, a unique Service Key is obtained through the registration of a service with jUDDI. As with each step a new authorization token is generated using the producer's initial registration information. In this successive step the Business Entity key, which was generated in the previous step, is used to assist with tying the two sets of registrant information together.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <save_service generic="2.0" xmlns="urn:uddie-org:api_v2">
   <authInfo>authToken:7ACBA3D0-C624-11DB-B794-9786F7ABC0B2</authInfo>
   <businessService businessKey="80604770-C623-11DB-B794-D20C319D25C5" serviceKey="">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <bindingTemplates>
     <bindingTemplate bindingKey="">
      <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="***">
        <instanceDetails>
         <overviewDoc>
          <overviewURL>***</overviewURL>
         </overviewDoc>
        </instanceDetails>
       </tModelInstanceInfo>
      </tModelInstanceDetails>
     </bindingTemplate>
    </bindingTemplates>
   </businessService>
  </save_service>
 </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <serviceDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <businessService businessKey="80604770-C623-11DB-B794-D20C319D25C5" serviceKey="81E48100-C624-11DB-B794-
B50CD54D7639">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <bindingTemplates>
     <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-
B50CD54D7639">
      <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo tModelKey="***">
        <instanceDetails>
         <overviewDoc>
          <overviewURL>***</overviewURL>
         </overviewDoc>
        </instanceDetails>
       </tModelInstanceInfo>
      </tModelInstanceDetails>
     </bindingTemplate>
    </bindingTemplates>
   </businessService>
  </serviceDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Continuing with the successful registration of our service we must obtain a binding key and a new service key, which assist with the registration of the actual service. In order to obtain this binding key we first needed to register the producer, obtain a new authorization token, and provide the producer's business and service keys.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <save_binding generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:9F85FE90-C625-11DB-B794-FEFC5EE8D774</authInfo>
      <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-B50CD54D7639">
        <description xml:lang="en">Service Producer Services</description>
        <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
      </bindingTemplate>
    </save_binding>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <bindingDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
    <bindingTemplate bindingKey="81E607A0-C624-11DB-B794-F458073DD97B" serviceKey="81E48100-C624-11DB-B794-B50CD54D7639">
      <description xml:lang="en">Service Producer Services</description>
      <accessPoint URLType="http">http://localhost:8080/myApp/services/MyWebClass</accessPoint>
    </bindingTemplate>
  </bindingDetail>
 </soapenv:Body>
</soapenv:Envelope>
```

Next we obtain a tModel key by registering the actual service. When registering the actual service, the authorization key we generate at each step identifies this service to its actual service producer.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Body>
  <save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
   <authInfo>authToken:9882AFC0-C626-11DB-B794-D5BF3565FF5A</authInfo>
   <tModel tModelKey="">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <overviewDoc>
     <description>***</description>     <overviewURL>platform:/resource/Servers/MyWebClass.wsdl</overviewURL>
    </overviewDoc>
    <identifierBag>
     <keyedReference tModelKey="***" keyName="***" keyValue="***" />
    </identifierBag>
    <categoryBag>
     <keyedReference tModelKey="***" keyName="***" keyValue="***" />
    </categoryBag>
   </tModel>
  </save_tModel>
 </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <tModelDetail generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <tModel authorizedName="Service Producer" operator="jUDDI.org" tModelKey="uuid:A78D6500-C626-11DB-B794-
96E880134AAD">
    <name>Service Producer</name>
    <description>Service Producer Services</description>
    <overviewDoc>
     <description>***</description>
<overviewURL>platform:/resource/Servers/MyWebClass.wsdl</overviewURL>
    </overviewDoc>
    <identifierBag>
     <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
    </identifierBag>
    <categoryBag>
     <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
    </categoryBag>
   </tModel>
  </tModelDetail>
 </soapenv:Body>
```

In addition to these steps listed other services were also registered so that more than one service existed within jUDDI. These other services are not shown here because their introduction would provide too much redundancy. The other service, not shown, is called *Service Certifier*.

Creating the publisherAssertion requires two publishers who make identical statements about each other. These identical statements may be within the same UDDI or across two different UDDIs. The format for this publisherAssertion is as follows:

Publisher Assertion From Service Certifier:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Publisher Assertion From Service Producer:
FromKey: 5FB39040-C623-11DB-B794-FB86F3278740
ToKey: 80604770-C623-11DB-B794-D20C319D25C5
TModel_Key (for Service Certifier): uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF
Key Name: Service Certifier
Key Value: parent-child

Notice that the two publisherAssertions are truly identical. The mechanism that prevents bogus claims is that each publisher assertion also requires an authorization token/key. The authorization token authenticates the Business Key. If a business key is used which does not authenticate with the authentication token a SOAP fault error is generated and the transaction is refused by jUDDI.

Following with the set of SOAP message examples below is the publisherAssertion transaction for service producer.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <add_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
      <authInfo>authToken:0A001290-C628-11DB-B794-944076723DD3</authInfo>
      <publisherAssertion>
        <fromKey>5FB39040-C623-11DB-B794-FB86F3278740</fromKey>
        <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
        <keyedReference tModelKey="uuid:3AE5D7C0-C626-11DB-B794-F8CC410B3ADF" keyName="Service Certifier" keyValue="parent-
child" />
      </publisherAssertion>
    </add_publisherAssertions>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
```

Finally, prospective service consumers who wish to identify complete publisherAssertions can do so by performing the following transaction.

SUBMIT

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_assertionStatusReport generic="2.0" xmlns="urn:uddi-org:api_v2">
      <authInfo>authToken:643D2D50-CAD4-11DB-AD50-9C9D409BE183</authInfo>
      <completionStatus>status:complete</completionStatus>
    </get_assertionStatusReport>
  </soapenv:Body>
</soapenv:Envelope>
```

RECEIVE

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <assertionStatusReport generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
   <assertionStatusItem completionStatus="status:complete">
    <fromKey>5FB39040-C623-11DB-B794-FB86F3278740</fromKey>
    <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
    <keyedReference keyName="Service Certifier" keyValue="parent-child" tModelKey="uuid:3AE5D7C0-C626-11DB-B794-
F8CC410B3ADF"/>
    <keysOwned>
     <toKey>80604770-C623-11DB-B794-D20C319D25C5</toKey>
    </keysOwned>
   </assertionStatusItem>
  </assertionStatusReport>
 </soapenv:Body>
</soapenv:Envelope>
```

Appendix B

TCSS 559 Development Environment for non-UWT Computers

Author: Philip Bonderud, UWT CSS Graduate Student
For: Dr. Sam Chung

This document explains how to install Tomcat, Eclipse, MySQL, JDK 5, and jUDDI on your home computer or laptop. This document assumes you are running Windows XP Professional and that you have at least one gig of RAM. If you have less than **1** gig of RAM, be aware that the more plug-ins you install the more RAM Eclipse will require. You may also want to close any unneeded programs while using Eclipse for Web Services development. If you are not running Windows XP Professional, you may need to seek out alternative solutions to configure your home environment on your own. When seeking alternative information, verify that it matches the programs listed in this document. Before scanning the internet for documentation on these programs, first explore whether your questions can be answered through help files that exist within the downloaded programs. Changes/modifications to this document appear on the last page of this document. The most recent changes appear in yellow. Finally, if these tools are new to you, please use the suggested passwords, paths, and filenames suggested in this document. The jUDDI installation is the most tedious process.

These files are available at www.service99.com through 12/31/07 and at the locations listed below. Open source products change version numbers frequently. Using versions other than these may produce unanticipated results which you must resolve on your own.

Tomcat-5.5.20 (http://tomcat.apache.org/download-55.cgi)
Binary Distributions, Core, using Windows Service Installer

MySQL-5.0.26-win32 (http://dev.mysql.com/downloads/mysql/5.0.html)
Windows (x86) file.

JDK 5.0 Update 10 (http://java.sun.com/javase/downloads/index.jsp)
http://java.sun.com/javase/downloads/index_jdk5.jsp
Includes Java Runtime Environment (JRE)
Use Windows Online Installation download

jUDDI0.9rc4 (http://ws.apache.org/juddi/releases.html)

Eclipse 3.2.1 http://www.eclipse.org
eclipse-SDK-3.2.1-win32.zip

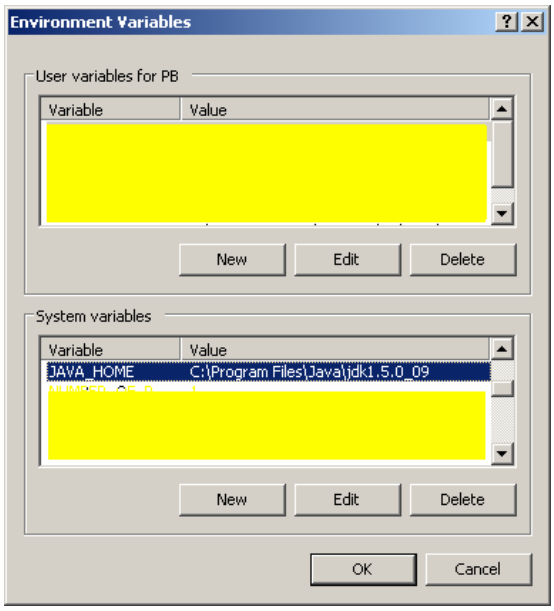
JDK 5.0

Download program, run installation, and accept default installation path. Install the full program.

After successful installation create/modify the JAVA_HOME environmental variable. Locate/Identify the full path to the new Java JDK 5.0 download. The default is C:\Program Files\Java\jdk1.5.0_10.

Click Start, Settings, Control Panel, System. On the Advanced tab click the Environment Variables button. Edit JAVA_HOME and add this to the end of the existing line (if it exists, otherwise create a new variable with this information and without the leading semicolon):

; C:\Program Files\Java\jdk1.5.0_10



In the above example, JAVA_HOME did not exist and was added.

Install Apache-Tomcat-5.5.20:

Install and read the documentation for the program. Throughout the Tomcat documentation you will see references to $CATALINA_HOME. When you install Tomcat, a folder will be created which will hold the various subfolders and files. The default folder settings for Tomcat is:

C:\Apache Software Foundation\Tomcat 5.5\blah blah blah

The root folder for Tomcat, which is also called $CATALINA_HOME, in this example is:

C:\Apache Software Foundation\Tomcat 5.5

The default settings for Tomcat's installation should work fine, however this scenario included installing the full program. The setup steps are:

- Run the downloaded Tomcat program
- For the first prompt simply to click Next.
- Agree/Accept the terms.

- Select Full install and click Next. A custom install (default) will probably work however within the environment used for this tutorial a full installation was selected.
- Accept the default destination folder: C:\Program Files\Apache Software Foundation\Tomcat 5.5 and click Next
- Leave the default port (8080) as is for now. Change the User Name and/or enter a password if you desire. You will need this information to access Tomcat's management tools. Entering a password is optional. Click Next.
- Verify the path to the Java installation you just performed. If you accepted Java's default settings you should see: C:\Program Files\Java\jre1.5.0_10. Click Install.
- Accept the final defaults and click Finish. Tomcat should start and the readme file may provide additional information for your reading pleasure.

Tomcat's toolbar icon looks like this:  Right clicking on it enables you to stop and restart Tomcat as needed. Usually, when Tomcat is running the center of the icon is a green rectangle. Tomcat's documentation states that its icon itself is not an accurate tool for determining whether the server is running. You will need to right click the icon to be sure.

Set your $CATALINA_HOME environmental variable to point to the root folder of where you installed Tomcat. If you are running more than one version of Tomcat on your home computer you may want to read up on $CATALINA_BASE. $CATALINA_BASE may be necessary, instead of $CATALINA_HOME, when multiple versions of Tomcat exist. It is up to you to determine whether this scenario applies to your system and this document assumes that only this version of Tomcat is used.

To set the environmental variable for this root Tomcat folder, click on Start, Settings, Control Panel, and then System. In the Advanced tab click on the Environment Variables button. At the bottom of the Environment Variables dialogue box (for System variables) create a new Variable named $CATALINA_HOME and give it a path to the root Tomcat folder. Using the scenario above my Variable value is C:\Program Files\Apache Software Foundation\Tomcat 5.5

Check your work: open a browser and enter the URI http://localhost:8080/ . If your installation was successful you should see the Tomcat logo, program version number, and some additional information. If you see a different set of information (other than an error) then you have another server running instead of Tomcat. Disable or uninstall the other server and try again.

Wherever this document references $CATALINA_HOME in this scenario it means "C:\Program Files\Apache Software Foundation\Tomcat 5.5." Therefore $CATALINA_HOME\blah means C:\Program Files\Apache Software Foundation\Tomcat 5.5\blah.

Install MySQL:

Download the Windows setup file named mysql-5.0.26-win32, unzip the item, and run the setup program. Select:

- Run the setup program, click Next
- Setup Type: Complete, click Next
- Accept default destination folder and click Install.
- Skip Sign-up (optional)

Configuring:

- Detailed Configuration, click Next
- Developer Machine, click Next
- Multifunctional Database, click Next
- MySQL Server Instance Configuration, InnoDB Tablespace Settings, accept default, click Next

- Decision Support (DSS)/OLAP (default) , click Next
- Enable TCP/IP Networking and Enable Strict Mode (yes – default on both) , click Next
- Standard Character Set (default) , click Next
- Install As Windows Service (default) and check the box "Include Bin Directory in Windows PATH, click Next
- Modify Security Settings (default) and enter the root password as: 123456 The effect of checking "Enabling root access from remote machines" has not yet been used and in this tutorial the box was left unchecked. Do not create an Anonymous Account.
- Execute

If configuration fails and you use a firewall, try disabling the firewall and re-executing.

Install MySQL JDBC Driver:

Unzip the MySQL file mysql-connector-java-5.0.4 into its own folder. Open the unzipped mysql-connector-java-5.0.4 folder and copy the jar file (named mysql-connector-java-5.0.4-bin) to $CATALINA_HOME\common\lib.

Install JUDDI:

Ignore all instructions that are posted on the website for JUDDI, they are not for this version. There are a lot of independent steps. If you miss a step, you will have to review the entire jUDDI installation process again and locate the error. ****Important**** If you are not following the naming conventions used in this tutorial you must amend the content below to match your MySQL details and it is up to you to discover what needs to be changed. Good luck! I recommend following these instructions for now and altering them later, after you are more comfortable with the environment. If you have any difficulty copying/pasting from the Word.doc version of this tutorial, most of the files which are created/edited are also available at www.service99.com\tutorials\program_files\jUDDI\shortcut.html. These files will be available on this site until 12/31/07.

1) Get jUDDI Program Files

Download the program file labeled juddi-0.9rc4 and drop it into C:\JUDDI. Unzip it using the default name. Once unzipped your jUDDI files will be located at C:\JUDDI\juddi-0.9rc4. The relevance of this is simply so that your path matches the paths shown in this document. Later we will use documentation and scripts that come with the downloaded file. The URI for JUDDI is: http://ws.apache.org/juddi/releases.html.

2) Copy/Paste juddi folder into Tomcat

Navigate to the JUDDI webapp folder. Within the JUDDI webapp folder copy the subfolder "juddi" and paste it into the $CATALINA_HOME\webapps folder. There is a JUDDI validation program which will help diagnose errors, but we're not ready to use it just yet.

3) Prepare MySQL for jUDDI

Return to the juddi-0.9rc4 folder and navigate to the mysql subfolder located at C:\JUDDI\juddi-0.9rc4\sql\mysql. Open the ReadMe file. If you have difficulty reading the ReadMe file, try opening it in a browser. This ReadMe file will provide additional information on the next step.

Covered in the ReadMe file are a couple scripts which will correctly prepare MySQL. Edit the ***insert_publishers.sql*** file to include the following values (for the email address enter any email address you wish):

VALUES ('juddi', 'juddi', 'you@domain.com', 'true', 'true'); If an error occurs when you run this script, chance are you have a typo. Below is an example of what your file should look like when finished (with your email address instead of mine). We will not make any changes to "create_database.sql."

```
USE juddi;

INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,EMAIL_ADDRESS,IS_ENABLED,IS_ADMIN)
VALUES ('juddi','juddi','poguywa@pbonderud.com','true','true');
```

Save your work and make sure that the filename did not change.

In the following steps “full_path” references the path to the named script files contained in this subfolder.

Click Start, Programs, MySQL, MySQL Server 5.0, and open the MySQL Command Line Client. Enter the password 123456 .

Enter:
\. full_path\create_database.sql

After pressing Enter again and running the script successfully we need to run the other script. Next run:
\. full_path\insert_publishers.sql

If you’ve followed the logic suggested in this document then you will have entered

\. C:\JUDDI\juddi-0.9rc4\sql\mysql\create_database.sql

followed by:

\. C:\JUDDI\juddi-0.9rc4\sql\mysql\insert_publishers.sql

Exit MySQL.

4) Create juddi.xml

Next, create a file called juddi.xml in $CATALINA_HOME\conf\Catalina\localhost and within it copy/paste the following. If you chose not to use the password noted in this document, you will need to edit the information below to reflect the password you chose. Please note, however, that password conflicts are problematic.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 6.0//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-web-jar.dtd">

<Context path="/juddi" docBase="juddi" debug="5" reloadable="true" crossContext="true">
<Logger className="org.apache.catalina.logger.FileLogger" prefix="localhost_juddiDB_log" suffix=".txt"
timestamp="true"/>
<!-- the Resource element will probably work better for you on Tomcat 5+ if you simply use a Resource only tag
with xml attributes as opposed to the nested ResourceParams and parameter elements -->
<Resource name="jdbc/juddiDB" auth="Container" type="javax.sql.DataSource" maxActive="100" maxIdle="30"
maxWait="10000" username="juddi" password="123456" driverClassName="org.gjt.mm.mysql.Driver"
url="jdbc:mysql://localhost:3306/juddi?autoReconnect=true" />
</Context>
```

5) Create log4j.properties

To enable log4j, you need to create the file $CATALINA_HOME\webapps\juddi\WEB-INF\classes\log4j.properties. In that file enter only this information:

```
#
# set the log file to $CATALINA_HOME\webapp\juddi\WEB-INF\{HOME}/juddi.log
# and not the ${PWD}/juddi.log
#
log4j.appender.LOGFILE.File=/opt/tomcat/logs/juddi.log
```

Save and close the file.

6) Edit juddi.properties

Amend the file juddi.properties at which is located at $CATALINA_HOME\webapps\juddi\WEB-INF\juddi.properties match the email address you used within insert_publishers.sql. This should match the domain name used for the email address within JUDDI's insert_publisher.sql script.

```
# The UDDI Operator Contact Email Address
juddi.operatorEmailAddress = poguywa@pbonderud.com
```

7) Edit server.xml

Next, edit the server.xml file which is located at $CATALINA_HOME\conf\server.xml and append the following to the file, immediately above the </Host> tag.

```
<Context path="/juddi" docBase="juddi"
debug="5" reloadable="true" crossContext="true">
<Logger className="org.apache.catalina.logger.FileLogger"
prefix="localhost_juddiDB_log" suffix=".txt"
timestamp="true"/>
<!-- the Resource element will probably work better for  you on Tomcat 5+ if you simply use a Resource only tag with xml attributes as opposed to the nested ResourceParams and parameter elements
-->

<Resource name="jdbc/juddiDB" auth="Container" type="javax.sql.DataSource" maxActive="100" maxIdle="30" maxWait="10000"
username="juddi" password="juddi" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/juddi?autoReconnect=true"/>

<!-- <Resource name="jdbc/juddi" auth="Container" type="javax.sql.DataSource"
maxActive="100" maxIdle="30" maxWait="10000" username="juddi" password="123456"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/juddi?autoReconnect=true"/>
 -->
</Context>
```

8) Edit juddi-user.xml

Finally, edit the juddi-users.xml file, located at $CATALINA_HOME\webapps\juddi\WEB-INF\juddi-users.xml and change the password for user juddi to 123456.

```
<user userid=”juddi” password=123456”>
```

9) Test Your Work:
Restart Tomcat. Right click the Tomcat icon, stop the service, then start the service. Open your Internet Explorer browser and enter the URI: http://localhost:8080/juddi.
Click on Validate and scroll for errors noted in red.

Juddi is happy now. "Juddi is happy" is how writers express that jUDDI is correctly configured. If you see any errors in your JUDDI validation, rework these instructions.

Install Eclipse 3.2.1

Obtain Eclipse. If using the Eclipse website to download the program choose these options. http://www.eclipse.org

Click the orange "Download Eclipse" button.

Click the icon shown in the screen snapshot below.

Next, pick a mirror and download the program. You must use version 3.2.1.

Unzip the program into C:\eclipse

Other than configuring a few settings within Eclipse itself this completes the installation process.

Establishing an SFTP Connection within Eclipse 3.2 to Repos using Plug-in

Eclipse Memory

Depending on the number of Eclipse plug-ins you install you may need to manually edit the Eclipse.ini file located in the root folder of your Eclipse installation. You may see this step repeated in other documentation but only one edit of this file is necessary. The values used below were sufficient for handling a larger number of plug-ins than those required for TCSS-460. The test environment for creating this documentation has 1.5 gigs of RAM. While creating a simple Web Service, several errors were resolved by increasing the minimal and maximum memory values shown below. You are free to experiment with these values and set them to anything that works for your environment. It is possible to set these values on the command line, when starting Eclipse, but it is up to you to discover how.

Open Eclipse.ini for editing. Within this file are the default minimum (Xms) and maximum (Xmx) memory settings for Eclipse. Increase the numeric values for these settings. Adding additional content, in addition to these three lines, may cause Eclipse to ignore the entire file. Settings that worked well for the author of this document were:

```
-vmargs
-Xms512m
-Xmx1024m
```

Save the Eclipse.ini file, close your editor, and make certain that the extension was not changed.

Eclipse Workspaces

Start Eclipse 3.2 and select or create a workspace. Eclipse will automatically create a workspace folder if you choose a workspace name that does not exist.

Eclipse Plug-ins and the Web Tools Platform

Web Services in Eclipse requires a plug-in called WTP (Web Tools Platform). Additional plug-ins will also be installed with this step. You are free to determine which are not absolutely necessary for this project or to simply grab all that this tutorial suggests.

- Open Eclipse
- Select Help – Software Updates – Find and Install
- Search for updates of the currently installed features, click Finish
- Select a mirror, click OK
- Accept all terms and allow all updates to take place

- If an error occurs during these updates, repeat these steps.
- Allow Eclipse to restart the workspace if prompted

Installing new plug-ins:

- Select Help – Software Updates – Find and Install
- Search for new features to install, click Next
- Select Callisto Discovery Site and click Finish
- Select a mirror and click OK
- Expand the list for Callisto Discovery Site
- As you select plug-ins, errors will appear at the top of the dialogue box which indicate that additional downloads are required. When this error appears click the right-side button: Select Required, so that all other dependent files are downloaded as well.
- As you add plug-ins the help list within Eclipse should also expand as well.
- If you have ample memory feel free to select all plug-ins except for C/C++ development. Otherwise start with Java Development, Web and J2EE Development (WTP) and expand your functionality later. It is up to you to learn how to use these features. Remember to use the "Select Required" button to resolve plug-in file dependency errors.
- Click Next
- Accept all terms, click Next
- Click Finish.
- If an error occurs during these updates, repeat these steps.
- Allow Eclipse to restart the workspace if prompted

You may also repeat the first part of this process to look for additional updates of the new installed features (optional).

Eclipse allows anyone to create plug-ins and tutorials exist which explain how to accomplish this. One point to mention is that nearly all tutorials are for earlier versions of Eclipse. As a graduate student creating this tutorial it has been incredibly frustrating to simultaneously learn and write/edit development environment documentation for Open Source programs. However, such is the nature of Open Source programs. The skill you develop to resolve these type of problems are skills which employers also desire.

Using Eclipse with Repos (SFTP Plugin)

When working with Repos you can use an FTP connection of your own and simply drag and drop files between your local computer and Repos. If you use this drag and drop method, expect the files to eventually collect garbage data and no longer validate or function properly. At least this was a situation encountered a couple years ago.

An alternative to manually dragging and dropping files is to enable Eclipse to push the files to Repos for you. The plug-in you need for this method is available through Jcraft.com.

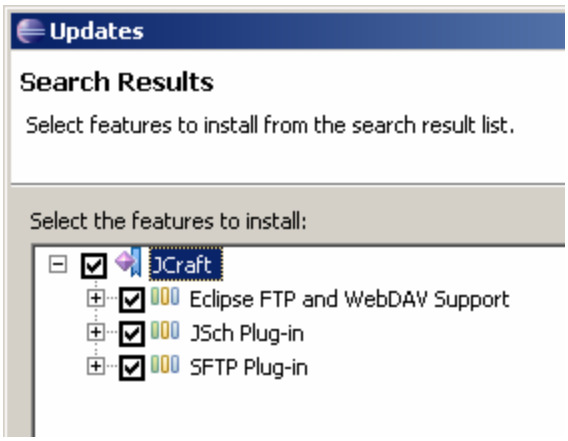
- Select Help – Software Updates – Find and Install
- Search for new features to install, click Next
- Click the New Remote Site button

A dialogue box will appear and prompt you to enter a name and URL:

- Name: JCraft
- URL: http://eclipse.jcraft.com/
- Click OK
- Click Finish

Eclipse will search the URI given and add the name of the plug-in site to its list of plug-in sites.

- After Eclipse locates the plug-ins, select all



- Click Next
- Accept all terms, click Next, click Finish.
- If prompted choose Install All.
- Allow Eclipse to restart when prompted, click Yes

Each time Eclipse starts you will be prompted to select a workspace. You only need to select a new workspace when you want to start fresh, from scratch, with no projects.

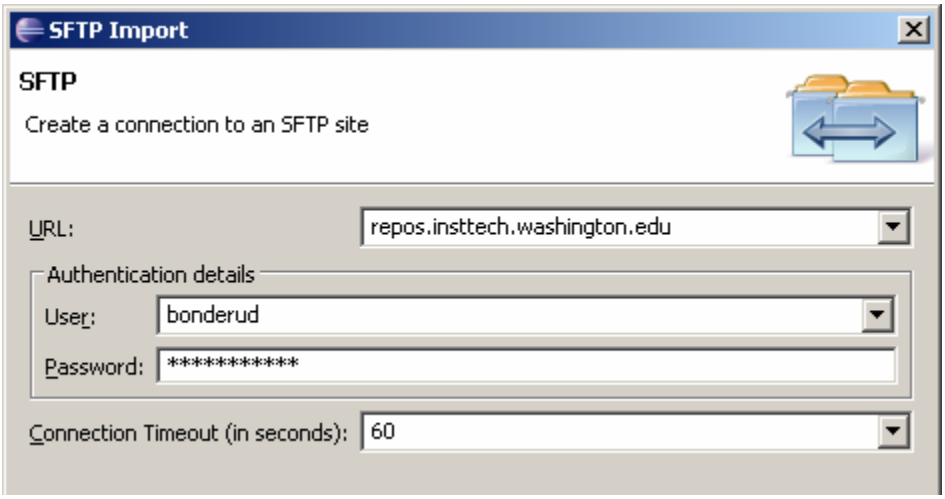
Using the JCraft SFTP Plug-In

Before we can use the SFTP Plug-in we need a project.

- File – New – Project – Web – Dynamic Web Project
- Name your project, this tutorial uses DynamicExampleProject
- Target runtime environment for Repos not known at the time of writing this tutorial, leave blank.
- Click Finish

Importing using SFTP

- File – Import – Other – SFTP – Next – Next
- Select "Create a New Site" - Next



- URL: repos.insttech.washington.edu
- User: replace “Bonderud” with your user ID name which you login to the insttech lab computers. Enter your password.
- Click Next
- A window will open which lists all of your files on Repos. Expand the folder listing and select items which you wish to import. Click Next.
- After Eclipse scans the files available in the selected folder(s) a second window will appear which prompts you to choose which files to import. Make your selection and click Finish.

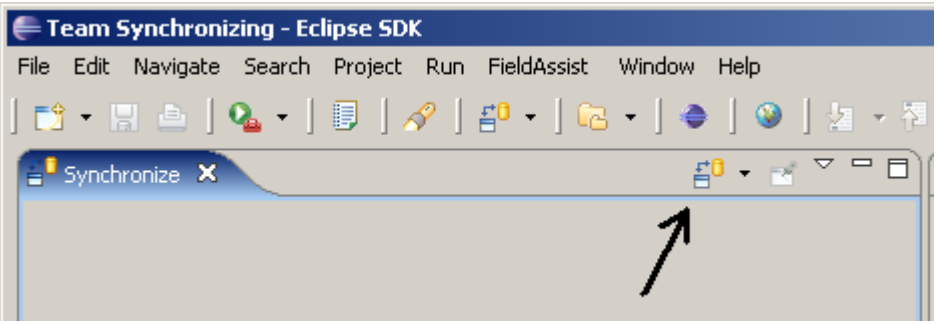
Synchronizing with a Remote Location

Once you have imported or exported a project of folder to a remote location the SFTP support remembers the association between the local resources and the remote location. This means that you can use the Synchronize View to move files back-and-forth between the local project and the remote location, as well as keep them synchronized (i.e. upload and download changes).

To create a Synchronization first open the Team Synchronizing perspective.

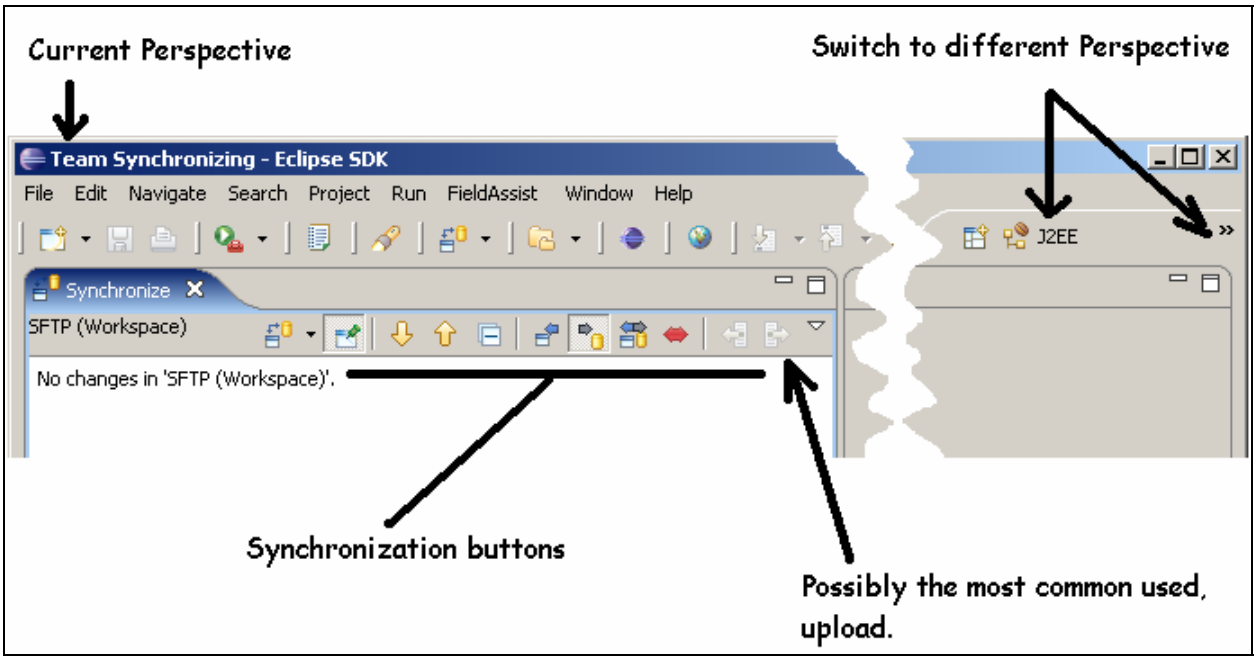
- Window – Open Perspective – Other
- Team Synchronizing

When the Synchronize View opens look for the  button, shown in the following figure, and click on it.



- Select SFTP and click Next
- Determine which files you wish to synchronize and click Finish

Switch between different views (called Perspectives) using the buttons near the upper right corner of your Eclipse work area (shown below) or using Window – Open Perspective. Which Perspective you are currently using is shown at the top left of your browser. Within the synchronization pane the buttons outlined will assist you in navigating and uploading your work to Repos.



Managing your Project Development in Eclipse

It may be helpful to enable Eclipse to automatically build your project for you each time you save your work.

- Click Project
- Click Build Automatically

Eclipse also offers a tool to publish to remote servers automatically, however this feature was not tested for this SFTP plugin or tutorial (enjoy).

- Click Window – Preferences
- Select Server
- Select Automatically Publish to Remote Server every (select time)
- You may need to configure an independent server within eclipse for the steps above.
- As mentioned, this is out of scope from this tutorial and may not be necessary.

Closing Notes

These are open source programs, therefore anticipate that some tweaking to these instructions may be necessary. Be wary of online tutorials. It is important to make sure that both the program version and environment for which an online author is speaking – are the same as yours.

Creating and Publishing Java Web Services Interfaces using Eclipse and the Web Tools Platform (WTP) 1.5

This document is a revision of Eclipse instructions [source: Alan "Re: How to Publish to My Own jUDDI Server?" http://dev.eclipse.org/newslists/news.eclipse.webtools/msg11863.html]. Follow the example names exactly as shown. Where an example specifies "MyApp" use this name for this tutorial. Later you can change this name to whatever you want. If you are unfamiliar with Tomcat, keep both the local version of Tomcat and Eclipse's internal Tomcat server set to port 8080. At times this will generate an error which will highlight which events Eclipse uses its internal Tomcat and the local Tomcat on your machine. Follow this tutorial and experiment with changes later. These steps may appear remedial, but when an error occurs they may help you narrow down the actual cause.
Before getting started it may be helpful to mention a few words about Eclipse out-of-memory errors. While attempting to launch your project in Eclipse, if you discover that Eclipse is generating out-of-memory errors there is a relatively painless remedy.

If you have not yet modified the Eclipse.ini file: Locate the root folder for Eclipse and open the Eclipse.ini file within a simple editor. There are 3 lines of code in this file with the following default values:

-vmargs
-Xms40m
-Xmx256m

After experimenting with these settings it was discovered that these settings eliminated Eclipse generated out-of-memory errors. Depending on the number of features/plug-ins you have active in your version of Eclipse, and the amount of RAM you have available, your default memory settings may be different.

-vmargs
-Xms512m
-Xmx1024m

Edit the Eclipse.ini file, save it, and restart Eclipse.

Next…

Eclipse Workspaces
Start Eclipse 3.2.1 and select or enter a new workspace name. Eclipse will automatically create a workspace folder if you choose a workspace name that does not exist.

1) Create a Tomcat server in the Eclipse workbench by following the instructions at the following link:
http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.jst.ws.axis.ui.doc.user/tasks/ttomcatserv.html

The following steps are done using the Eclipse IDE along with the WTP plugin:

2) Create a "Dynamic Web Project" by doing the following inside of Eclipse:
- File->New->Dynamic Web Project

- Give the project an appropriate name and select the "Finish" button. We'll call it "myApp" as a reference in this document.

3) Create your Java class with the public methods that you will be using as your Web Service Interfaces - this is called bottom up web design. Do this as follows:

- RIGHT click on the "myApp" project and select
New->Class

- Give an appropriate package name (e.g. my.org.name) and class name (e.g. MyWebClass) and select the "Finish" button.

- Finish coding up the public methods for the "MyWebClass" class.
For this example use the following simple Java file:

```
package my.org.name;
/**
 * @author Phil Bonderud
 * This simple Web Service simply doubles a name and/or
 * counts the number of letters in a name.
 */
public class MyWebClass {
	/**
	 * @param name Any name
	 * @return String A phrase followed by two names.
	 */
	public String appendName(String name){
		return "Seeing double? "+name+" "+name;
	}

	/**
	 * @param name Any name
	 * @return int The number of letters in the name
	 */
	public int countLetters(String name){
		return name.length();
	}
}
```

- For this exercise ‘build automatically’ is selected. To enable this feature click Project then select Build Automatically.

- Make sure your project has all the required jar files in its "Java Build Path", i.e. do the following:
RIGHT click on your project and select "Properties", then under "Java Build Path" add all the required jar files needed the compile your class.

- Make sure you build your project so that your class is available in the output folder (e.g. build/classes/...), i.e. do the following to create the class files:
Project->Build Project

4A) RIGHT click on the "MyWebClass.java" [myApp/srs/my.or.name/MyWebClass.java] file and do the following to create the WSDL and the test ~~client~~ service:

- Web Services->Create Web Service

- In the "Web Services" window, move the upper slider to the top position (Test Service) then select Next->Next.

-On the Test Web Service Window click Launch. If you have a local copy of Tomcat running, shut it down, wait 1 – 2 minutes, then click Launch. The Web Services Explorer will open within a new browser window in order for you to test your service (view WSDL Binding Details). Within the left side Navigator window, click on any of the methods. After clicking on a method the main window changes to enable you to Invoke a WSDL operation. In the space provided enter an appropriate parameter and click "go." In the Status window below the results of your transaction will appear – check for correctness. To view the underlying SOAP messages click "Source" within the Status window.

- Click Finish

4B) RIGHT click on the "MyWebClass.java" [myApp/srs/my.or.name/MyWebClass.java] file and do the following to create the WSDL and the test service:
- Web Services->Create Web Service

- In the "Web Services" window, move the upper slider to the Start Service position and the bottom slider to the Test Service position and then select Next->Next.

-On the Test Web Service Window click Launch. If you have a local copy of Tomcat running, shut it down, wait 1 – 2 minutes, then click Launch. The Web Services Explorer will open within a new browser window in order for you to test your service (view WSDL Binding Details). Within the left side Navigator window, click on any of the methods. After clicking on a method the main window changes to enable you to Invoke a WSDL operation. In the space provided enter an appropriate parameter and click "go." In the Status window below the results of your transaction will appear – check for correctness. To view the underlying SOAP messages click "Source" within the Status window.

- Click Finish

5) Once the "Web Services Explorer" window come up, you can test your interfaces. Some users have found that testing the interfaces using this mechanism may result in peculiar behaviors. For these users, what works better is to migrate your project to Tomcat by doing the following on the target machine:

5.1 – If running, stop Tomcat on the target machine/platform.

5.2 - Create a directory with the SAME name as your project, in this case it's "myApp", under the following directory -> TOMCAT_HOME/webapps which should result in something like TOMCAT_HOME/webapps/myApp

5.3 - Copy all contents under the "WebContent" folder for your project from your development machine (e.g. C:\EclipseWorkspace\myApp\WebContent*.*) to the target machine under TOMCAT_HOME/webapps/myApp.

5.4 - If your Eclipse output folder is defaulted to the "build" folder, then copy the "classes" folder for your project from your development machine (e.g. C:\EclipseWorkspace\myApp\build\classes) to the target machine under TOMCAT_HOME/webapps/myApp/WEB-INF.

5.5 - You should now have a similar directory structure as the following on your target machine:

--> TOMCAT_HOME/webapps/myApp/META-INF
--> TOMCAT_HOME/webapps/myApp/WEB-INF
--> TOMCAT_HOME/webapps/myApp/WEB-INF/classes
--> TOMCAT_HOME/webapps/myApp/wsdl

5.6 - Eclipse can create the client application project which you can use to test your Web Services, it will have the same name as your project's name with the word "Client" appended to it, for this discussion it would be "myAppClient". The code for the client can be used to create your own client classes to access your Web Service Interfaces, so you can make good use of it.

We are going to take steps similar to those in step **4B** with a couple differences.

-RIGHT click on the "MyWebClass.java" [myApp/srs/my.or.name/MyWebClass.java] -Select Web Services->Create Web Service
- In the "Web Services" window, move the upper slider to the Start Service position and the bottom slider to the Test Service position and then select Finish.

Eclipse will create the myAppClient files (including some JSP files) and directory structure automatically. Eclipse will also open a Web Services Test Client window in which you may test your Web Service using Eclipse. Move on to step 8.7 to learn how to run this client within your local version of Tomcat.

5.7 - Repeat steps **5.2-5.5** and migrate the client application [including the JSP files located in the sampleMyWebClassProxy folder] (e.g. myAppClient) to TOMCAT/webapps/myAppClient too.

6) Start Tomcat on the target machine and Tomcat should recognize your new web applications located under the TOMCAT_HOME/webapps directory. You can access and test the "myApp" Web Services by running the "myAppClient" JSP that was created by Eclipse. The typical URL for the client JSP is as follows (notice that it uses your application name and class name):

http://localhost:8080/myAppClient/sampleMyWebClassProxy/TestClient.jsp

7) If you make changes to your "MyWebClass" using Eclipse copy the "MyWebClass.class" file from your development machine (e.g. C:\EclipseWorkspace\myApp\build\classes\myPackageName\...\MyWebClass.class) to the target machine (e.g. TOMCAT_HOME/webapps/myApp/WEB-INF/classes/myPackageName/.../MyWebClass.class). Restart Tomcat on the target machine and retest until everything is in order and your Web Services Interfaces are working as required.

8) At this point, using your Java class, you have successfully created your Web Services Interfaces and their WSDL document - the WSDL document lives under the TOMCAT_HOME/webapps/myApp/wsdl directory and can be accessed using the browser via a URL similar to the following:
http://localhost:8080/myApp/services/MyWebClass?wsdl

Using the following URL will show you some other WSDL interfaces created by Eclipse:
http://localhost:8080/myApp/services

Publishing the WSDL document into the UDDI registry...

Next we want to publish the "myWebClass" WSDL into a UDDI registry/server. In this example we will be using the jUDDI server that was installed on our target machine. If the jUDDI server was installed correctly, its main page should be at a URL similar to the following:
http://localhost:8080/juddi

By looking in the "controller.jsp" file that resides under the TOMCAT_HOME/webapps/juddi/console directory, we can see that the "inquiry" URL is something like "http://localhost:8080/juddi/inquiry"; - we will need this URL later in the wizard.

*Note: Your local version of Tomcat must be running in order for these next steps to execute successfully.

1) Bring up Eclipse and do:
run->Launch the Web Services Explorer

2) In the Web Services Explorer window, click on the "UDDI Main" link.

3) For the "Inquiry URL" field enter the jUDDI inquiry URL that you got from the "controller.jsp" file, e.g.
http://localhost:8080/juddi/inquiry

4) Give the jUDDI server an appropriate name for the "Registry Name" field, e.g. "My jUDDI", then click on the "Go" button.

5) In the "Registry Details" screen, under "Other Actions", you can click on the "Add To Favorites" link to add your jUDDI registry server to the drop down list (for future usage).

6) In the "Registry Details" screen, under "Other Actions", click on the "Publish" link to bring up the "Publish" screen. Use the following options to publish the "MyWebApp" services:

- Publish -> Service Interface
- Publication format -> simple
- Publish URL -> http://localhost:8080/juddi/publish
- User ID -> myUserId (this is the user ID that you have allowed to use the jUDDI database)
- Password -> <I don't believe jUDDI is currently using this field, so you can enter anything>
- WSDL URL -> platform:/resource/myApp/WebContent/wsdl/MyWebClass.wsdl
- Name -> My Application
- Description -> My Web Class Services

For the "WSDL URL" field, you must use the browse button to select the myApp project and Eclipse should be able to find the WSDL document from that project and import it into that field.

Notice that the "Publish URL" is the URL for the jUDDI server's "publish" address (i.e. where we're publishing the WSDL document) and NOT where the MyWebClass service interface lives (i.e. not http://localhost:8080/myApp/services/MyWebClass).

Searching for the Service...

1) Bring up Eclipse and do:
run->Launch the Web Services Explorer

2) In the Web Services Explorer window, click on the "UDDI Main" link.

3) For the "Inquiry URL" field enter the jUDDI inquiry URL that you got from the "controller.jsp" file, e.g. http://localhost:8080/juddi/inquiry or use the drop down list if you had saved it already. Click the “Go” button.

4) In the "Registry Details" screen, under "Other Actions", click on the "Find" link to bring up the "Find" screen.

5) Use the following options in the "Find" screen to find your service:

- Search for -> Service Interfaces
- Type of search -> simple
- Name -> My Application (or whatever "Name" you used in step 6 above)

Click on the "Go" button and the information for the "MyWebClass" services should be displayed. Notice that you can "Edit" certain fields such as the "WSDL URL" field. You might want to use the actual URL for the Web Services such as http://my.host.machine.name:8080/myApp/services/MyWebClass?wsdl (in this tutorial: http://localhost:8080/myApp/services/MyWebClass?wsdl)instead of the default one listed in the MySql database, e.g. "platform:/resource/myApp/WebContent/wsdl/MyWebClass.wsdl

Appendix C

TCSS 460 Development Environment for non-UWT Computers

Author: Philip Bonderud, UWT CSS Graduate Student
For: Dr. Sam Chung

This document explains how to install Tomcat, Eclipse, MySQL, and JDK 5 on your home computer or laptop. This document assumes you are running Windows XP Professional and that you have at least one gig of RAM. If you have less than **1** gig of RAM, be aware that the more plug-ins you install the more RAM Eclipse will require. This may be more of an issue for TCSS-559 (Web Services) since several plug-ins are required. If you are not running Windows XP Professional, you may need to seek out alternative solutions to configure your home environment on your own. When seeking alternative information, verify that it matches the programs listed in this document. Before scanning the internet for documentation on these programs, first explore whether your questions can be answered through help files that exist within the downloaded programs.

These files are available at www.service99.com and at the locations listed below. Open source products change version numbers frequently. Using versions other than these may produce unanticipated results which you must resolve on your own.

Tomcat-5.5.20 (http://tomcat.apache.org/download-55.cgi)
Binary Distributions, Core, using Windows Service Installer

MySQL-5.0.26-win32 (http://dev.mysql.com/downloads/mysql/5.0.html)
Windows (x86) file.

JDK 5.0 Update 9 (http://java.sun.com/javase/downloads/index.jsp)
http://java.sun.com/javase/downloads/index_jdk5.jsp
Includes Java Runtime Environment (JRE)
Use Windows Online Installation download

Eclipse 3.2.1 http://www.eclipse.org
eclipse-SDK-3.2.1-win32.zip

Parts of this document were derived from outdated sources online and modified to work within this environment [13].

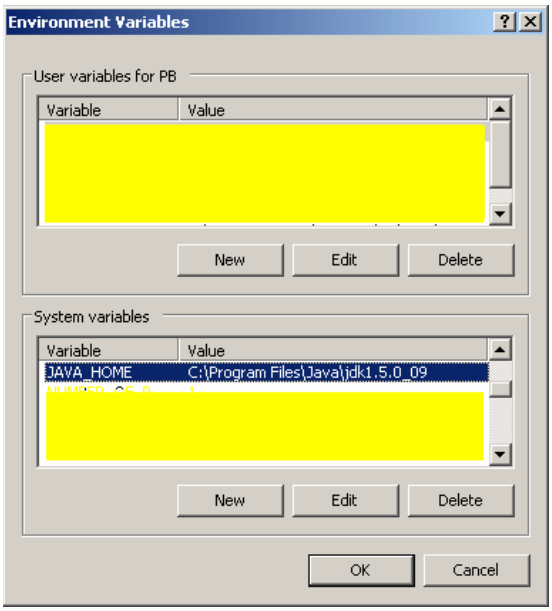
JDK 5.0

Download program, run installation, and accept default installation path. Install the full program.

After successful installation create/modify the JAVA_HOME environmental variable. Locate/Identify the full path to the new Java JDK 5.0 download. The default is C:\Program Files\Java\jdk1.5.0_09.

Click Start, Settings, Control Panel, System. On the Advanced tab click the Environment Variables button. Edit JAVA_HOME and add this to the end of the existing line (if it exists, otherwise create a new variable with this information and without the leading semicolon):

; C:\Program Files\Java\jdk1.5.0_09



In the above example, JAVA_HOME did not exist and was added.

Install Apache-Tomcat-5.5.20:

Install and read the documentation for the program. Throughout the Tomcat documentation you will see references to $CATALINA_HOME. When you install Tomcat, a folder will be created which will hold the various subfolders and files. The default folder settings for Tomcat is:

C:\Apache Software Foundation\Tomcat 5.5\blah blah blah

The root folder for Tomcat, which is also called $CATALINA_HOME, in this example is:

C:\Apache Software Foundation\Tomcat 5.5

Set your $CATALINA_HOME environmental variable to point to the root folder of where you installed Tomcat. If you are running more than one version of Tomcat on your home computer you may want to read up on $CATALINA_BASE. $CATALINA_BASE may be necessary, instead of $CATALINA_HOME, when multiple versions of Tomcat exist. It is up to you to determine whether this scenario applies to your system and this document assumes that only this version of Tomcat is used.

To set the environmental variable for this root Tomcat folder, click on Start, Settings, Control Panel, and then System. In the Advanced tab click on the Environment Variables button. At the bottom of the Environment Variables dialogue box (for System variables) create a new Variable named $CATALINA_HOME and give it a path to the root Tomcat folder. Using the scenario above my Variable value is C:\Program Files\Apache Software Foundation\Tomcat 5.5 .

Check your work: open a browser and enter the URI http://localhost:8080/ . If your installation was successful you should see the Tomcat logo, program version number, and some additional information. If you see a different set of information (other than an error) then you have another server running instead of Tomcat. Disable or uninstall the other server and try again.

Wherever this document references $CATALINA_HOME in this scenario it means “C:\Program Files\Apache Software Foundation\Tomcat 5.5.” Therefore $CATALINA_HOME\blah means C:\Program Files\Apache Software Foundation\Tomcat 5.5\blah.

Install MySQL:

Download the Windows setup file named mysql-5.0.26-win32, unzip the item, and run the setup program. Use the default settings and install the complete package as a service. Within setup, assign the root password as “123456.”

Install Eclipse 3.2.1

Obtain Eclipse. If using the Eclipse website to download the program choose these options. http://www.eclipse.org

Click the orange “Download Eclipse” button.

Click the icon shown in the screen snapshot below.

Next, pick a mirror and download the program. You must use version 3.2.1.

Unzip the program into C:\eclipse

Other than configuring a few settings within Eclipse itself this completes the installation process.

Establishing an SFTP Connection within Eclipse 3.2 to Repos using Plug-in

Eclipse Memory

Depending on the number of Eclipse plug-ins you install you may need to manually edit the Eclipse.ini file located in the root folder of your Eclipse installation. You may see this step repeated in other documentation but only one edit of this file is necessary. The values used below were sufficient for handling a larger number of plug-ins than those required for TCSS-460. The test environment for creating this documentation has 1.5 gigs of RAM. While creating a simple Web Service, several errors were resolved by increasing the minimal and maximum memory values shown below. You are free to experiment with these values and set them to anything that works for your environment. It is possible to set these values on the command line, when starting Eclipse, but it is up to you to discover how.

Open Eclipse.ini for editing. Within this file are the default minimum (Xms) and maximum (Xmx) memory settings for Eclipse. Increase the numeric values for these settings. Adding additional content, in addition to these three lines, may cause Eclipse to ignore the entire file. Settings that worked well for the author of this document were:

```
-vmargs
-Xms512m
-Xmx1024m
```

Save the Eclipse.ini file, close your editor, and make certain that the extension was not changed.

Eclipse Workspaces

Start Eclipse 3.2 and select or create a workspace. Eclipse will automatically create a workspace folder if you choose a workspace name that does not exist.

Eclipse Plug-ins and the Web Tools Platform

Web Services in Eclipse requires a plug-in called WTP (Web Tools Platform). Additional plug-ins will also be installed with this step. You are free to determine which are not absolutely necessary for this project or to simply grab all that this tutorial suggests.

- Open Eclipse
- Select Help – Software Updates – Find and Install
- Search for updates of the currently installed features, click Finish
- Select a mirror, click OK

- Accept all terms and allow all updates to take place
- If an error occurs during these updates, repeat these steps.
- Allow Eclipse to restart the workspace if prompted

Installing new plug-ins:

- Select Help – Software Updates – Find and Install
- Search for new features to install, click Next
- Select Callisto Discovery Site and click Finish
- Select a mirror and click OK
- Expand the list for Callisto Discovery Site
- As you select plug-ins, errors will appear at the top of the dialogue box which indicate that additional downloads are required. When this error appears click the right-side button: Select Required, so that all other dependent files are downloaded as well.
- As you add plug-ins the help list within Eclipse should also expand as well.
- If you have ample memory feel free to select all plug-ins except for C/C++ development. Otherwise start with Java Development, Web and J2EE Development (WTP) and expand your functionality later. It is up to you to learn how to use these features. Remember to use the "Select Required" button to resolve plug-in file dependency errors.
- Click Next
- Accept all terms, click Next
- Click Finish.
- If an error occurs during these updates, repeat these steps.
- Allow Eclipse to restart the workspace if prompted

You may also repeat the first part of this process to look for additional updates of the new installed features (optional).

Eclipse allows anyone to create plug-ins and tutorials exist which explain how to accomplish this. One point to mention is that nearly all tutorials are for earlier versions of Eclipse. As a graduate student creating this tutorial it has been incredibly frustrating to simultaneously learn and write/edit development environment documentation for Open Source programs. However, such is the nature of Open Source programs. The skill you develop to resolve these type of problems are skills which employers also desire.

Using Eclipse with Repos (SFTP Plugin)

When working with Repos you can use an FTP connection of your own and simply drag and drop files between your local computer and Repos. If you use this drag and drop method, expect the files to eventually collect garbage data and no longer validate or function properly. At least this was a situation encountered a couple years ago.

An alternative to manually dragging and dropping files is to enable Eclipse to push the files to Repos for you. The plug-in you need for this method is available through Jcraft.com.

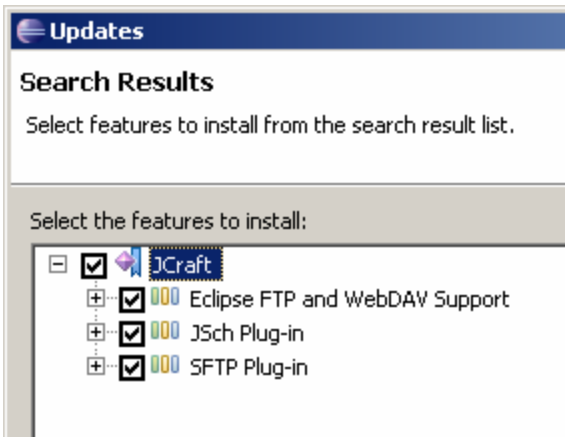
- Select Help – Software Updates – Find and Install
- Search for new features to install, click Next
- Click the New Remote Site button

A dialogue box will appear and prompt you to enter a name and URL:

- Name: JCraft
- URL: http://eclipse.jcraft.com/
- Click OK
- Click Finish

Eclipse will search the URI given and add the name of the plug-in site to its list of plug-in sites.

- After Eclipse locates the plug-ins, select all



- Click Next
- Accept all terms, click Next, click Finish.
- If prompted choose Install All.
- Allow Eclipse to restart when prompted, click Yes

Each time Eclipse starts you will be prompted to select a workspace. You only need to select a new workspace when you want to start fresh, from scratch, with no projects.

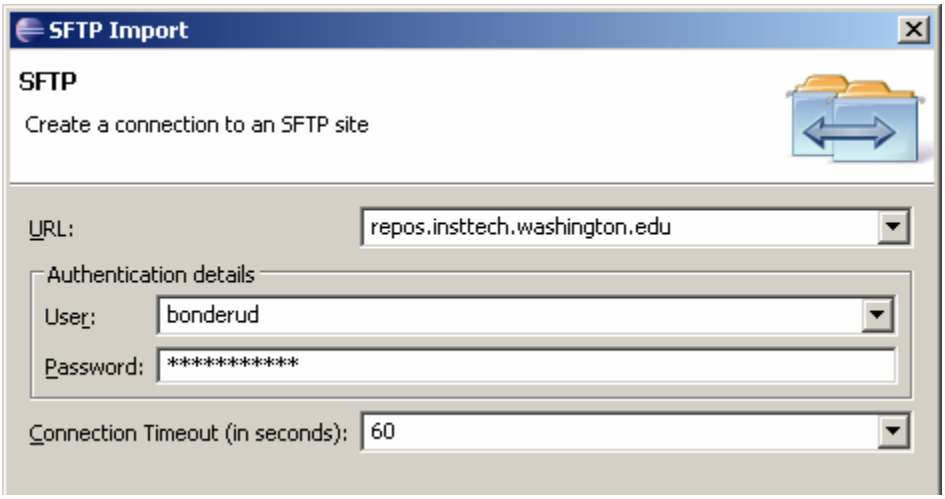
Using the JCraft SFTP Plug-In

Before we can use the SFTP Plug-in we need a project.

- File – New – Project – Web – Dynamic Web Project
- Name your project, this tutorial uses DynamicExampleProject
- Target runtime environment for Repos not known at the time of writing this tutorial, leave blank.
- Click Finish

Importing using SFTP

- File – Import – Other – SFTP – Next – Next
- Select "Create a New Site" - Next



- URL: repos.insttech.washington.edu
- User: replace “Bonderud” with your user ID name which you login to the insttech lab computers. Enter your password.
- Click Next
- A window will open which lists all of your files on Repos. Expand the folder listing and select items which you wish to import. Click Next.
- After Eclipse scans the files available in the selected folder(s) a second window will appear which prompts you to choose which files to import. Make your selection and click Finish.

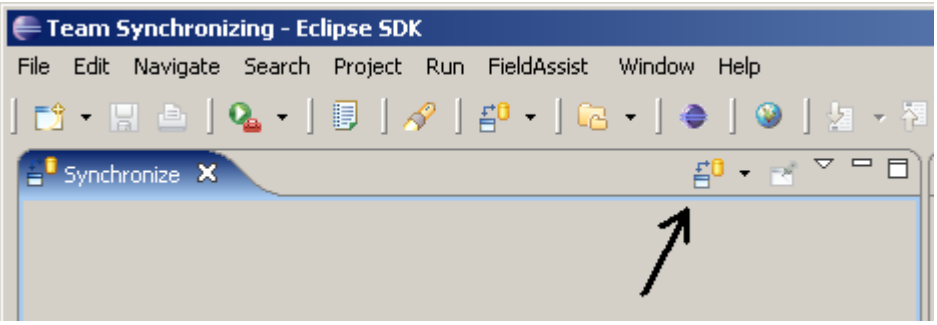
Synchronizing with a Remote Location

Once you have imported or exported a project of folder to a remote location the SFTP support remembers the association between the local resources and the remote location. This means that you can use the Synchronize View to move files back-and-forth between the local project and the remote location, as well as keep them synchronized (i.e. upload and download changes).

To create a Synchronization first open the Team Synchronizing perspective.

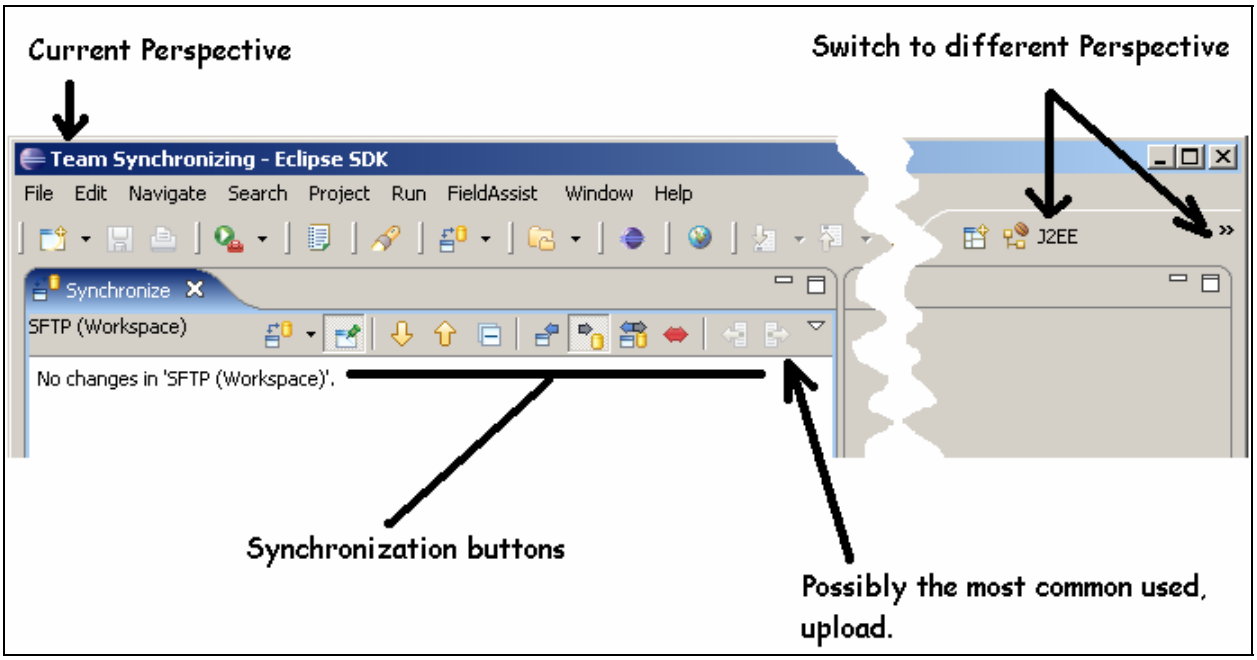
- Window – Open Perspective – Other
- Team Synchronizing

When the Synchronize View opens look for the  button, shown in the following figure, and click on it.



- Select SFTP and click Next
- Determine which files you wish to synchronize and click Finish

Switch between different views (called Perspectives) using the buttons near the upper right corner of your Eclipse work area (shown below) or using Window – Open Perspective. Which Perspective you are currently using is shown at the top left of your browser. Within the synchronization pane the buttons outlined will assist you in navigating and uploading your work to Repos.



Managing your Project Development in Eclipse

It may be helpful to enable Eclipse to automatically build your project for you each time you save your work.

- Click Project
- Click Build Automatically

Eclipse also offers a tool to publish to remote servers automatically, however this feature was not tested for this SFTP plugin or tutorial (enjoy).

- Click Window – Preferences
- Select Server
- Select Automatically Publish to Remote Server every (select time)
- You may need to configure an independent server within eclipse for the steps above.
- As mentioned, this is out of scope from this tutorial and may not be necessary.

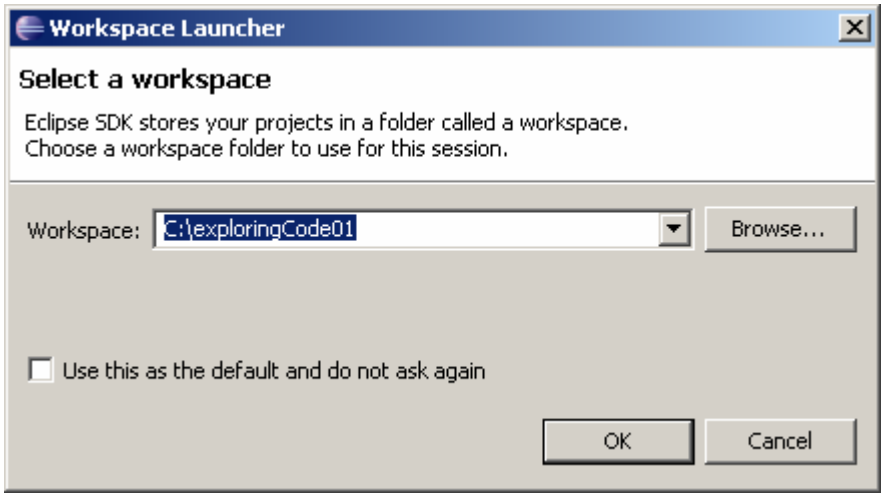
Closing Notes

These are open source programs, therefore anticipate that some tweaking to these instructions may be necessary. Be wary of online tutorials. It is important to make sure that both the program version and environment for which an online author is speaking – are the same as yours.

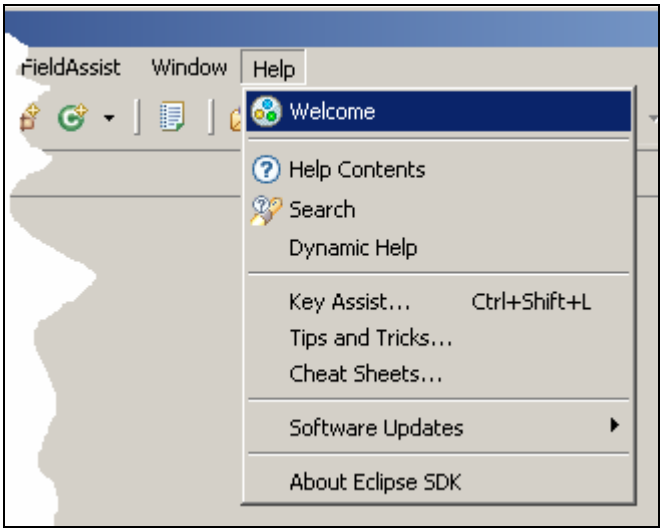
Creating a Simple Eclipse Project

On your non-UWT computer, Eclipse will start by prompting you to select a Workspace. Each workspace can hold any number of projects. You can select a default workspace and "tell" Eclipse not to ask again or you can create unique workspaces to accommodate various projects. If the workspace path does not exist Eclipse will create it.

For example, your first experimentation with Eclipse may simply be to explore code. You could create the workspace C:\exploringCode01 as shown and Eclipse will create the folder for you. How you structure your workspaces is entirely up to you.



When Eclipse first opens a "Welcome" view appears along with several round icons. Exploring these tutorials from Eclipse is helpful if you have not used Eclipse 3.2 before. Close the Welcome window for now. Whenever you want to retrieve this welcome window, simply click Help and select Welcome as shown below.



- Select Window/Preferences/Server/Installed Runtimes
- Click the Add button
- Expand the Apache list of servers and highlight Apache Tomcat v5.5. Click Next
- The Tomcat installation directory is the same is $CATALINA_HOME. Use the browse button to avoid typos and browse to C:\Program Files\Appache Software Foundation\Tomcat 5.5
- For your JRE you really want JDK 5. View this list of available options. If JDK 5 appears in the list, highlight it. Otherwise click the Installed JRE button.
    - Click Add

- Click Browse and browse to C:\Program Files\Java\jdk1.5._09 (or where ever you installed your JDK. and highlight the folder named jdk1.5._09
- Click OK
- Select jdk1.5.0_09 and click OK
- You should now be back in the Tomcat Server dialogue box. Again, under JRE scroll to view the list of available options and highlight jdk1.5.0_09
- Click Finish
- Click OK to close the Preferences window.

It's up to you, but you may want to enable Eclipse to Build Automatically, each time you click Save. To turn this feature on or off, click Project and then add or remove the check mark adjacent to "Build Automatically."

Perhaps the best source of information for your work will be the help which is available through your Eclipse program. Many tutorials exist at the Eclipse website and elsewhere online, however you may also discover that these tutorials do not match Eclipse 3.2's environment and they also contain errors. It's up to you. Figuring out these mistakes is helpful but time consuming.

A Simple Example

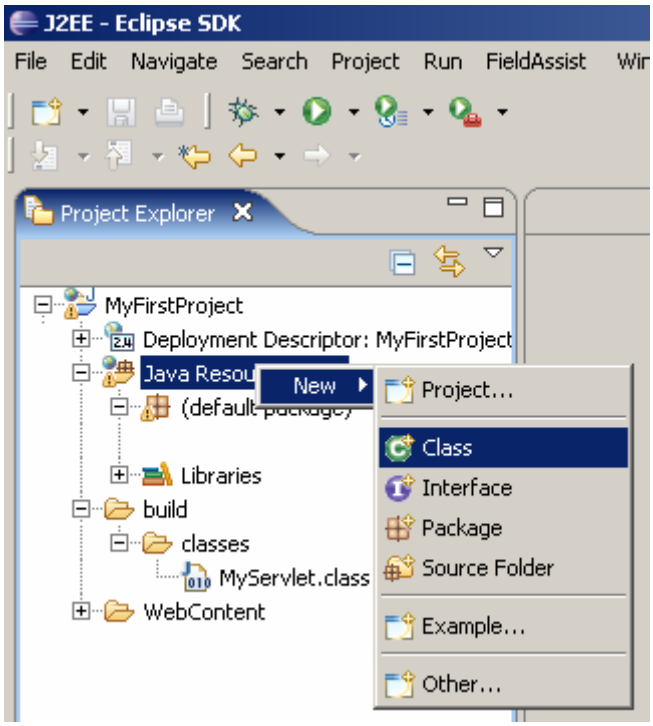
- Click File\New\Project\Other
- Scroll down and expand the available options for Web
- Highlight Dynamic Web Project and click Next
- Give your project the name MyFirstProject. If later you wish to look at the files outside of Eclipse, you can find them at C:\workspaceName\ProjectName
- The Target Runtime should be set to Apache Tomcat v5.5
- Leave the Configuration setting as is.
- Click Finish.
- Agree to all license agreements.
- Yes to switch Perspectives (read about Perspectives in Eclipse Help).

The type of perspective currently in use is stated at the left upper most corner of Eclipse.

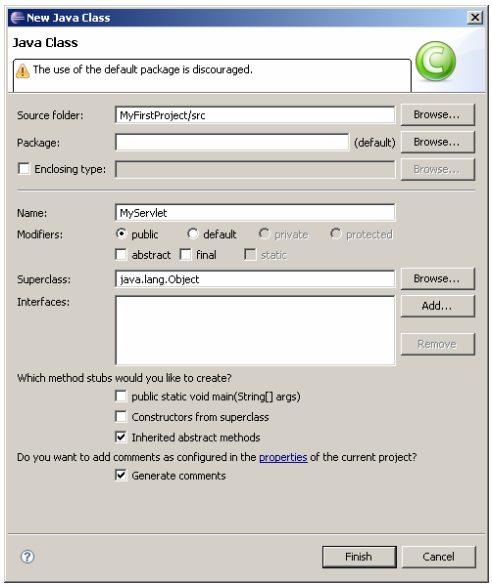
On the left side of your perspective is the Project Explorer. Expand the list for MyFirstProject. Notice that Eclipse automatically creates a set of files and file structure for you.

Hello World

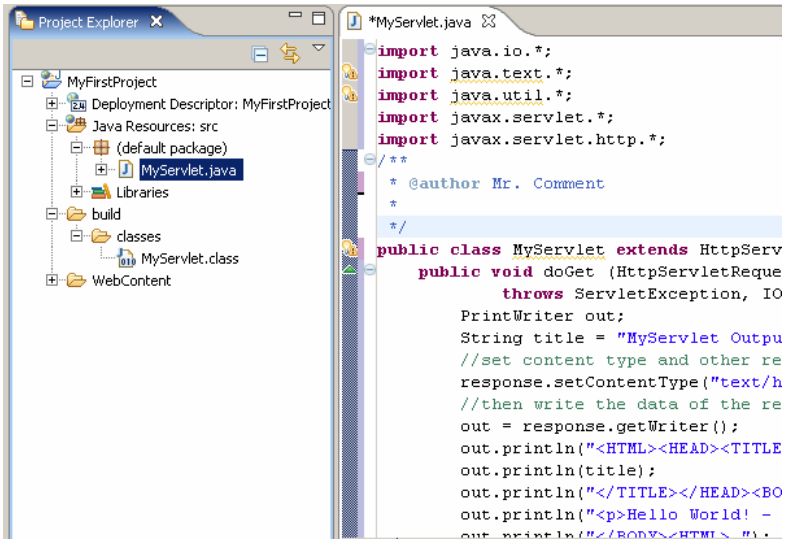
Right-Click on Java Resources: src, select New, select Class.



Name the new Java class “MyServlet”, accept the defaults for this first exercise, and enable Generate Comments (see below). Click Finish when done.



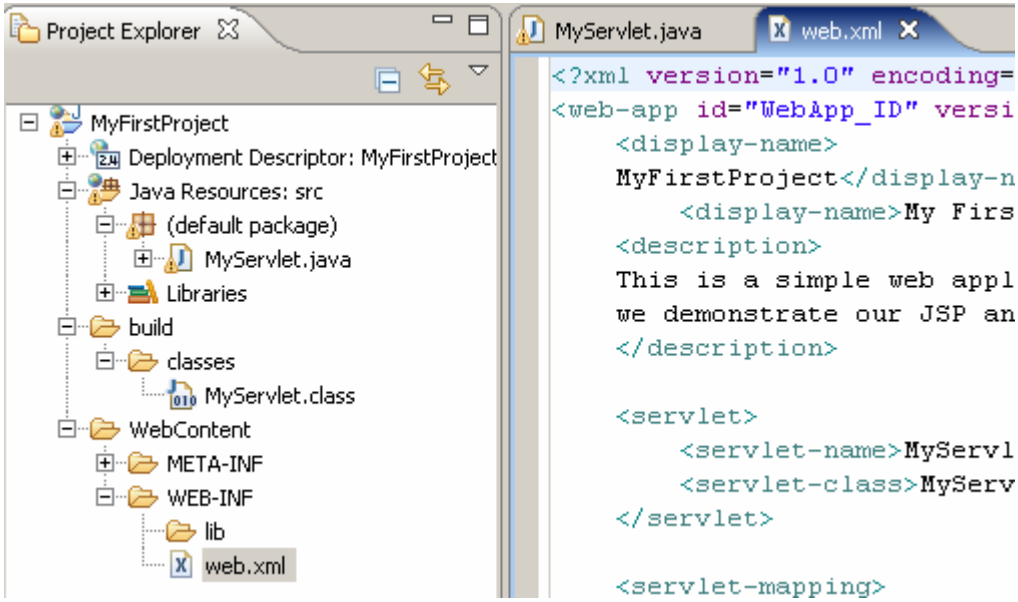
Highlight your new class so that the source code appears in the center window. Eclipse will create stub Javadoc for your code and make the final task of documenting your work much easier.



Copy this code into your MyServlet.java file and replace "Mr. Comment" with your own name.

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
/**
 * @author Mr. Comment
 *
 */
public class MyServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response )
                    throws ServletException, IOException {
     PrintWriter out;
     String title = "MyServlet Output";
     //set content type and other response header fields first
     response.setContentType("text/html");
     //then write the data of the response
     out = response.getWriter();
     out.println("<HTML><HEAD><TITLE> ");
     out.println(title);
     out.println("</TITLE></HEAD><BODY>");
     out.println("<p>Hello World! - This is the first servlet example");
     out.println("</BODY><HTML> ");
     out.close();
  }
}
```

Next we need to edit the Web.xml file. Within the Project Explorer pane expand WebContent and then expand WEB-INF to reveal the Web.xml file. Double click this file to bring it up for editing in the center window as shown below:



Now replace all of the information between the <web-app> </web-app> tags with the following:

```
<display-name>My First Servlet!</display-name>
<description>
        This is a simple web application in which
        we demonstrate our JSP and JavaBean demo.
</description>

<servlet>
   <servlet-name>MyServlet</servlet-name>
   <servlet-class>MyServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>MyServlet</servlet-name>
   <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

When you are finished your XML file will have some type of header, an opening <web-app> tag, the information in the box above, and a closing </web-app> tag.

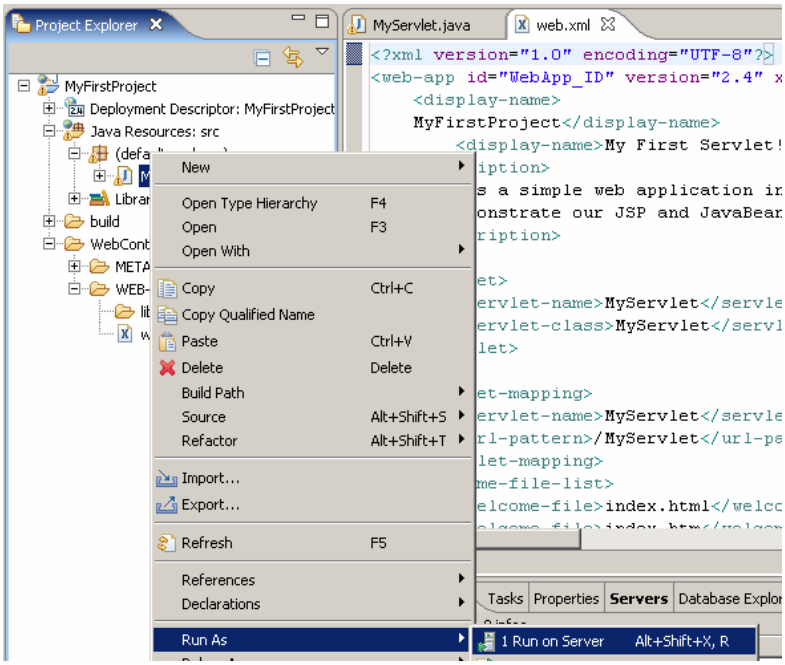
Ignore the warnings at the bottom for this example and save your work (File\Save All)

Running the Servlet within Eclipse

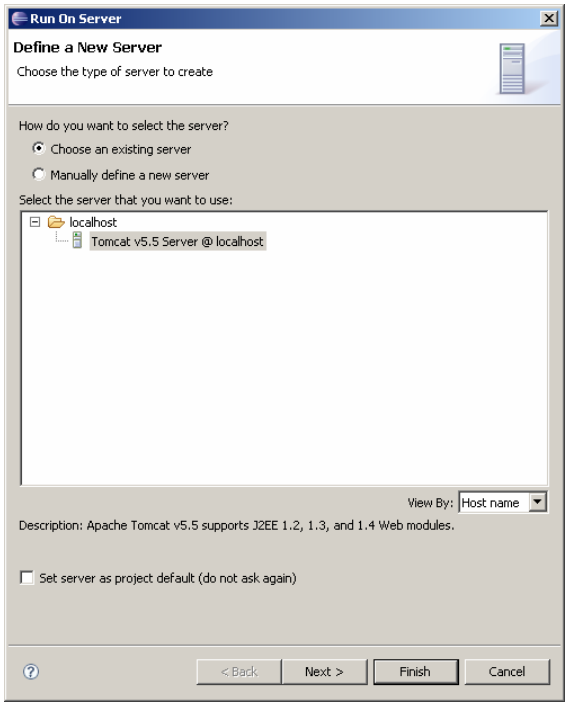
If Tomcat is up and running on your machine you must turn it off. Later you can assign one set of port numbers for Tomcat on your local machine and another set of port numbers for Tomcat use within Eclipse, but for now just turn off your local version.

Locate the Tomcat icon  located in the lower right corner of your screen. Right click this icon and select "Stop Service." It takes about 2 minutes for Eclipse to recognize that Tomcat is turned off.

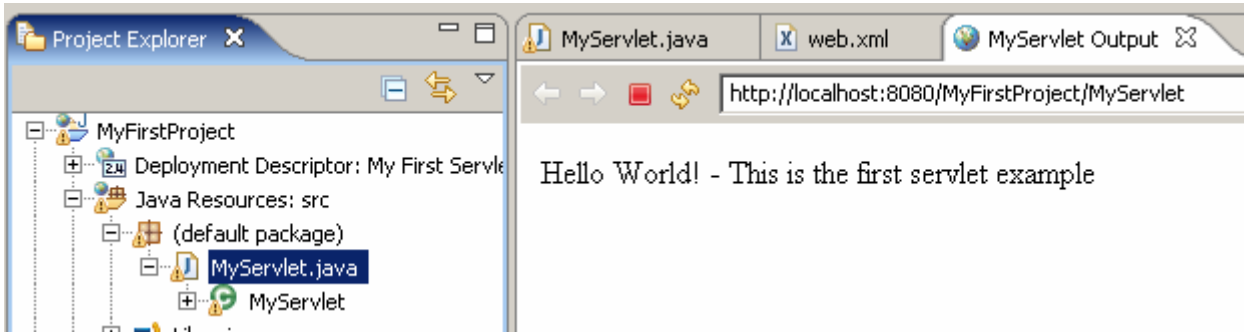
Next, within the Project Explorer pane, right-click the MyServlet.java file which you recently created. Scroll down the menu and hover your mouse over "Run As", and then select "1 Run on Server"



A "Run on Server" box may appear. Verify that your server is selected (Tomcat 5.5) and click finish.

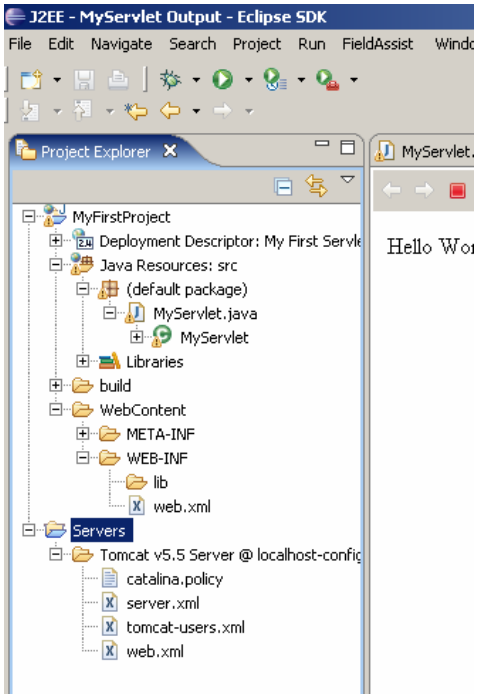


If you have set up everything correctly you will see the following message in the center pane of Eclipse:

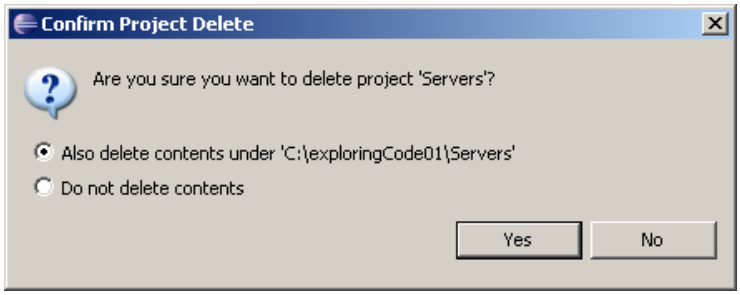


End of Hello World Servlet tutorial, now for a comment or two.

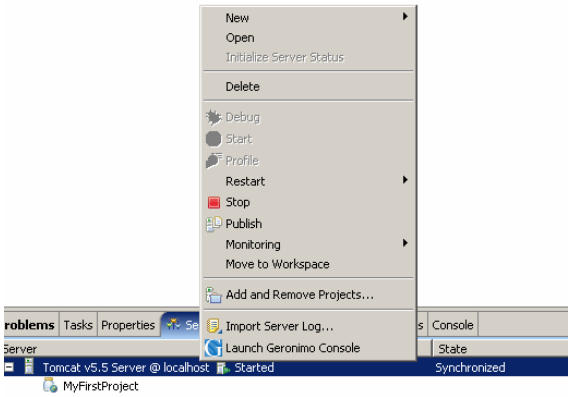
Look at the Project Explorer pane and notice that Eclipse added a new item: Server. Eclipse also created an association between your project and this internal server. There may be a time when you need to manually create this association in advance.



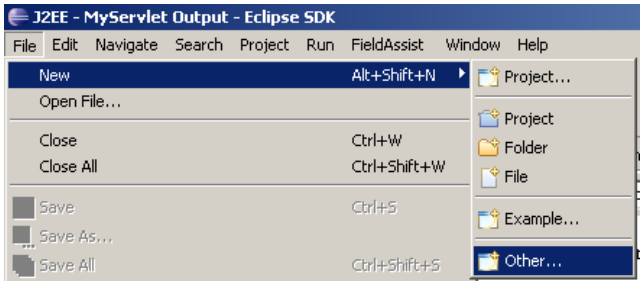
Highlight the server shown. Delete it along with its contents (Eclipse will prompt you with this question).



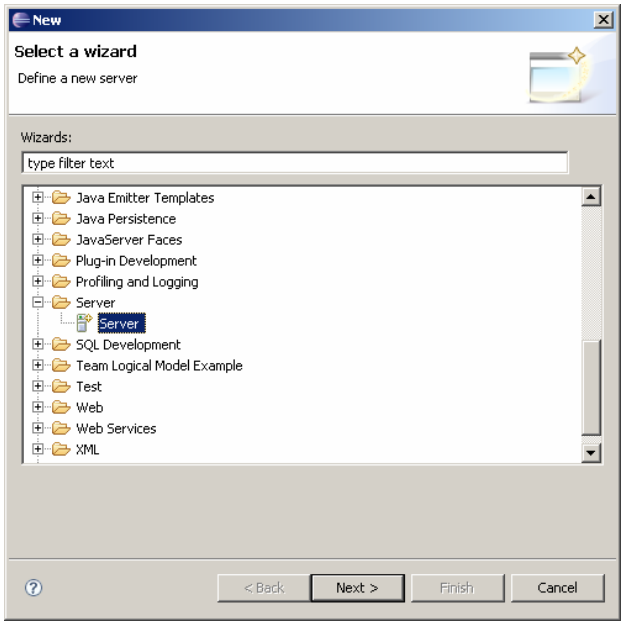
Next, along the bottom of your perspective stop the server and delete it also (right clicks bring up the menu).



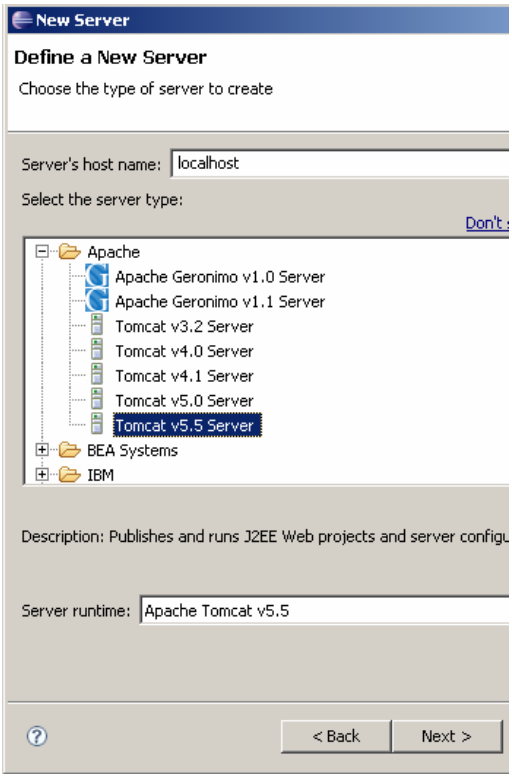
Now, let's manually recreate the server. Using the upper menu, click File\New\Other



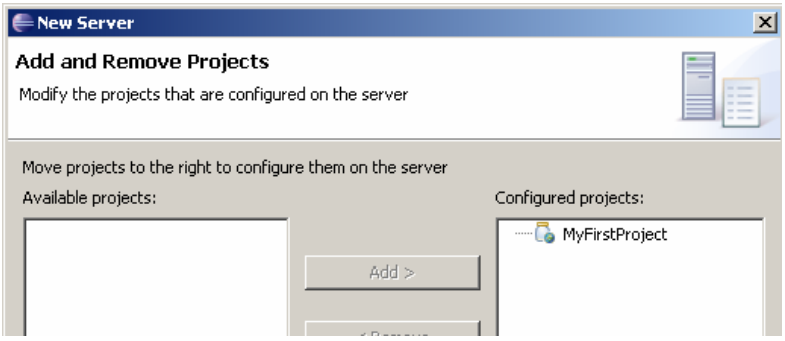
From the list that appears, expand the list of choices under Server and click Next.



Select Tomcat v5.5 Server and click Next.



On the next window, shown below, you will be asked to associate projects with this server. Highlight your project name and click the Add button so that it looks like the image below. Click Finish.



Notice that both areas where we deleted server information are again repopulated. If you right click on MyServlet.java again and select “Run As”, “1 Run on Server”, Eclipse will start its internal server and run your project.

There is a lot to learn with Eclipse but it is an excellent tool. IBM offers proprietary development software based on Eclipse, such as Webphere. Taking the time to get to know Eclipse is worth the effort.