



# T320 E-business technologies: foundations and practice

## Block 3 Part 4 Activity 2: Publishing to and accessing UDDI

Prepared for the course team by Neil Simpkins

---

Introduction	1
UDDI data structures	2
Publishing with Eclipse	3
Accessing the UDDI	3
Publishing entries	8
Publishing with the jUDDI Console	15
The Console	15
Authenticating	16
Publishing	18
Finding UDDI entries	24
Accessing UDDI services	26
'Other Actions' list options	26
Web Service Wizard	26
WSDL Main	29
Summary	32

---

### Introduction

You are now going to publish a description of the 'Hello' web service to a UDDI directory. There are two different tools that you can use to do this:

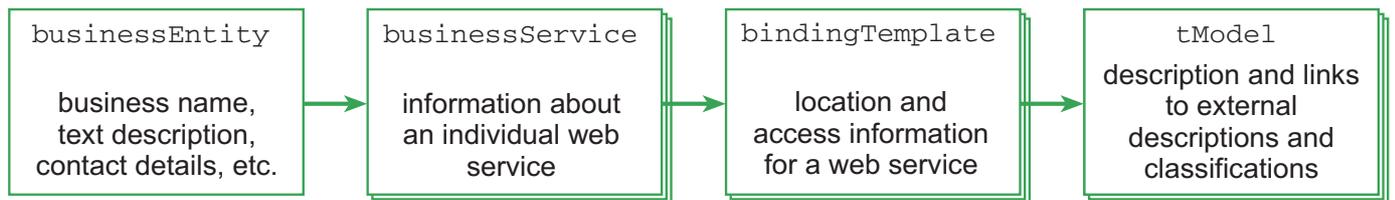
- 1 Eclipse and the WTP
- 2 the jUDDI Console web pages in your jUDDI server account at the OU.

Of these the first is perhaps more straightforward, and has the advantage that the same WTP facility can be used to publish to UDDI as may be used to search a UDDI directory and then also to access any web services via their descriptions.

Here I am first going to show you how to use Eclipse and the WTP to publish descriptions; then I shall outline the same process using the Axis2 web pages. Note that if you were personally to install jUDDI under Tomcat then you would probably want to use the web pages to validate the installation and test that it was working, rather than using Eclipse.

## UDDI data structures

The data structures that we want to create for a web service description are illustrated in Figure 1.



**Figure 1** UDDI data structures (repeated from Part 4)

To publish the 'Hello' service, for example, we want to create a single business description that can be linked to any number of service descriptions. Then, for the service, we can create one or more bindings linking the web service to a tModel that points to the service's WSDL.

However, the tool you will be using, the Eclipse WTP Web Services Explorer, has simplified this view somewhat. Using the Explorer, there is no direct access to tModels; thus in some respects the Explorer's view is incomplete.

Yet whilst the Explorer does not recognise all the standard UDDI data structures, it is able to access and manipulate a UDDI – which itself, of course, does embody the standard UDDI structures. So the view the Explorer provides is mapped onto underlying UDDI data structures. As you might expect, these are actually implemented as tables in a database<sup>1</sup>.

The Explorer's view means that we need only deal with three entities:

- 1 Businesses, which are roughly the same concept as a businessEntity
- 2 Services, which are very similar to a businessService
- 3 Service Interfaces, which link a WSDL to a tModel UUID. You can think of this as something close to a tModel.

Both Services and Service Interfaces can name a WSDL file location and can thus be used by Eclipse to create a client, as you saw earlier in the block.

Some accounts of using the Explorer suggest 'publishing' a web service based solely on the publishing of a single Service Interface. However, I prefer to remain closer to the spirit of UDDI and to publish a fuller set of descriptions, which in this case are composed of a Business that is linked to one or more Services. When we publish a service, the Explorer will access the WSDL document and will also automatically publish a Service Interface based on this information. Publishing a fuller set of descriptions will facilitate searching for the service and provide greater support for others wanting to contact the publishing organisation, for example.

For your information, the Explorer provides other 'linking' facilities:

- Referenced Services are links that can be defined to services published by other organisations.

<sup>1</sup> jUDDI can be used with a wide range of database systems, but MySQL has been used in this course.

- Publisher Assertions allow, for example, an organisation representing a set of businesses to link them together (for instance, if a large organisation owns several companies).

You will not be using these facilities in this course.

The WTP is heavily influenced by development by IBM, and has specific support for WSIL, which I have not investigated in this block. WSIL is described in the Eclipse help documentation as follows:

Web Services Inspection Language (WSIL) is a service discovery mechanism that is an alternative to UDDI as well as complementary to UDDI. ... WSIL allows you to go directly to the service provider and ask for the services it provides.

...

For more information on the Web Services Inspection Language specification, refer to [www.ibm.com/developerworks/webservices/library/ws-wsilspec.html](http://www.ibm.com/developerworks/webservices/library/ws-wsilspec.html).

For this reason you will see that the Explorer handles both WSDL and WSIL. I shall not be investigating WSIL in this course, but if you are interested there is substantial information on the Internet. Later in this activity you will use the WSDL Explorer (and WSDL Main); there is a similar facility for employing WSIL documents.

## Publishing with Eclipse

Before you start to publish entries to UDDI, a word (or several) of caution: it is very easy to publish entries, but it can be more difficult to keep track of the entries you have published and what they contain. So, you should publish only a single business entry and keep your other services, tModels and bindings to the minimum required for your deployed service(s). Entries can be deleted if you wish.

Also you must adhere very carefully to the naming conventions proposed here – so use your OU identifier as a prefix in naming a business, services and any other entries you create.

## Accessing the UDDI

Start Eclipse if it is not already running. The first step is to start the Web Services Explorer, so select this from the Run menu (Figure 2).

After a short time you will see that the Web Services Explorer opens in the main 'editing' area inside Eclipse. Click on the 'UDDI Main' node in the Navigator and you will be presented with the 'Open Registry' page, which can be used to access any UDDI (Figure 3). At this stage you may well wish to maximise the Web Services Explorer window in Eclipse, as this is used independently of other windows.

Notice that there is a range of six buttons at the top right-hand corner of the Web Services Explorer. These support rapid access to common operations that I shall cover shortly. If you hover the mouse over any button then you will see a 'tool-tip' message appear that describes the function of that button. The leftmost two buttons (left- and right-pointing arrows) operate in the same way as a typical web browser's 'Back' and 'Forward' functions in moving through the historical set of pages that have been visited.

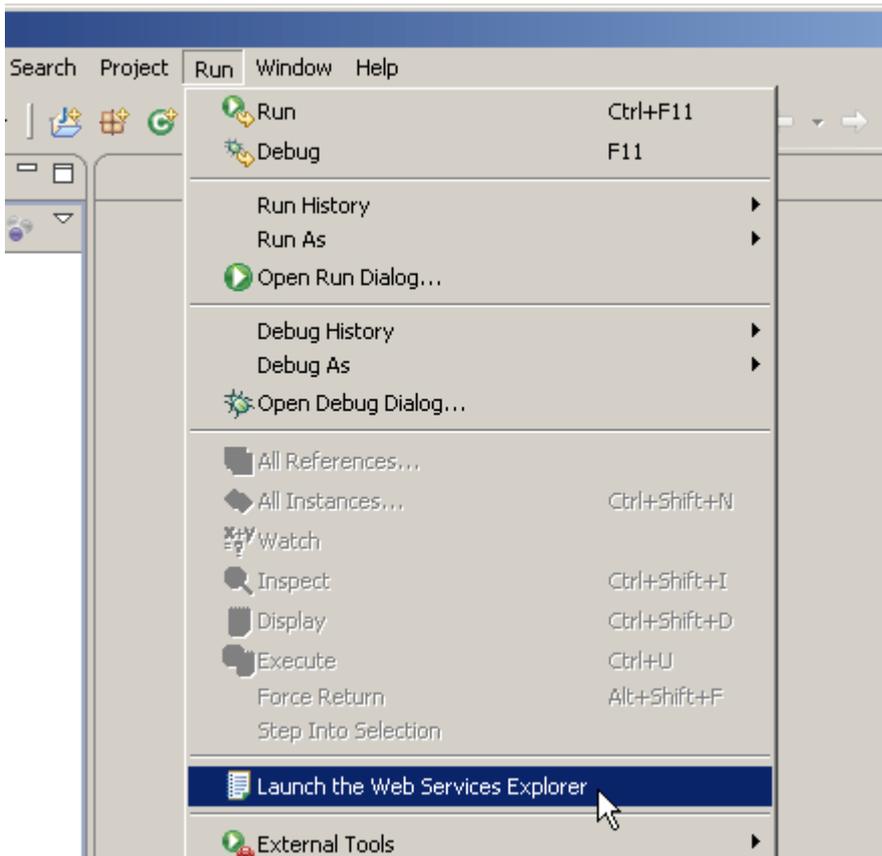


Figure 2 Launch the Web Services Explorer option on the Run menu

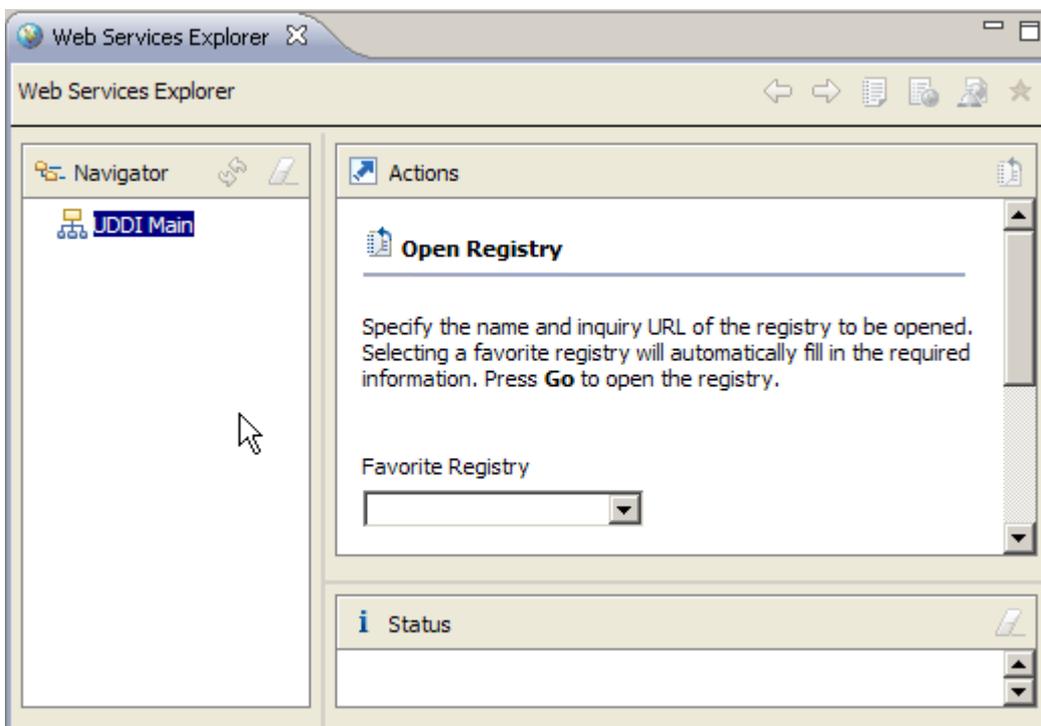


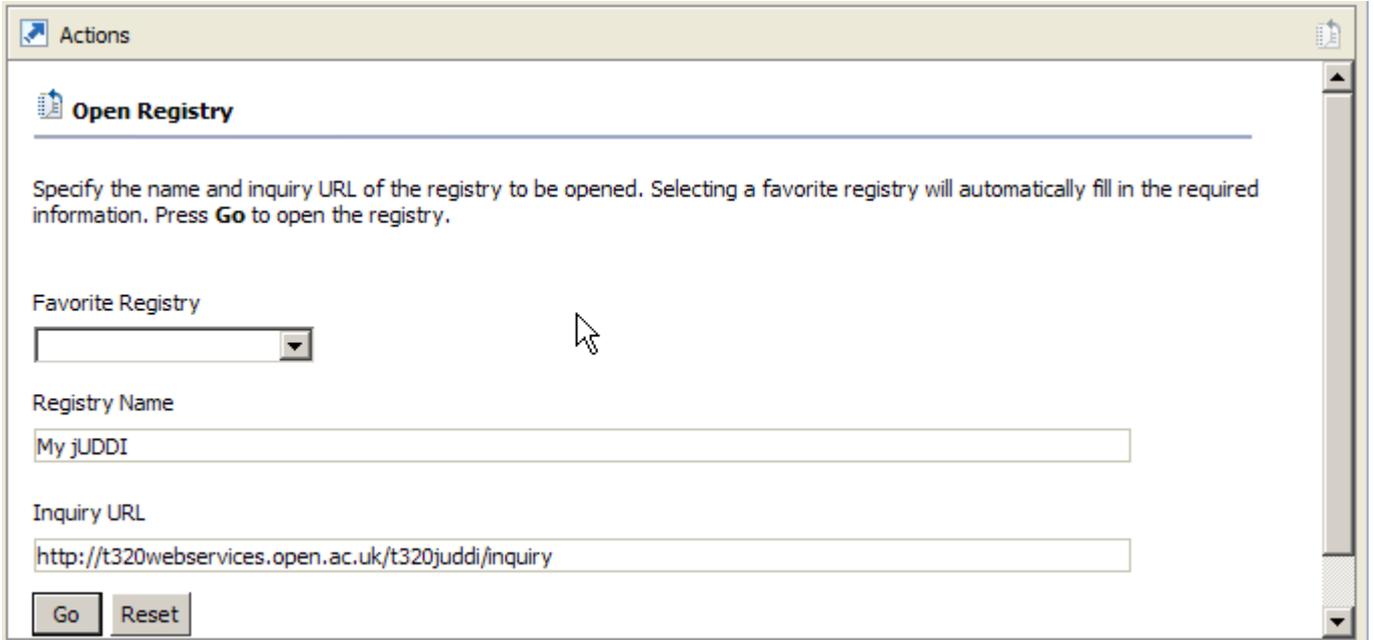
Figure 3 Web Services Explorer pane in Eclipse

The next step is to open the UDDI so that you can start interacting, either by searching for entries or by publishing. To open your own UDDI you need to enter the appropriate URL.

Earlier, in the introduction to T320 server accounts, I examined the jUDDI server pages. The URL for the jUDDI at the University and its enquiry interface is:

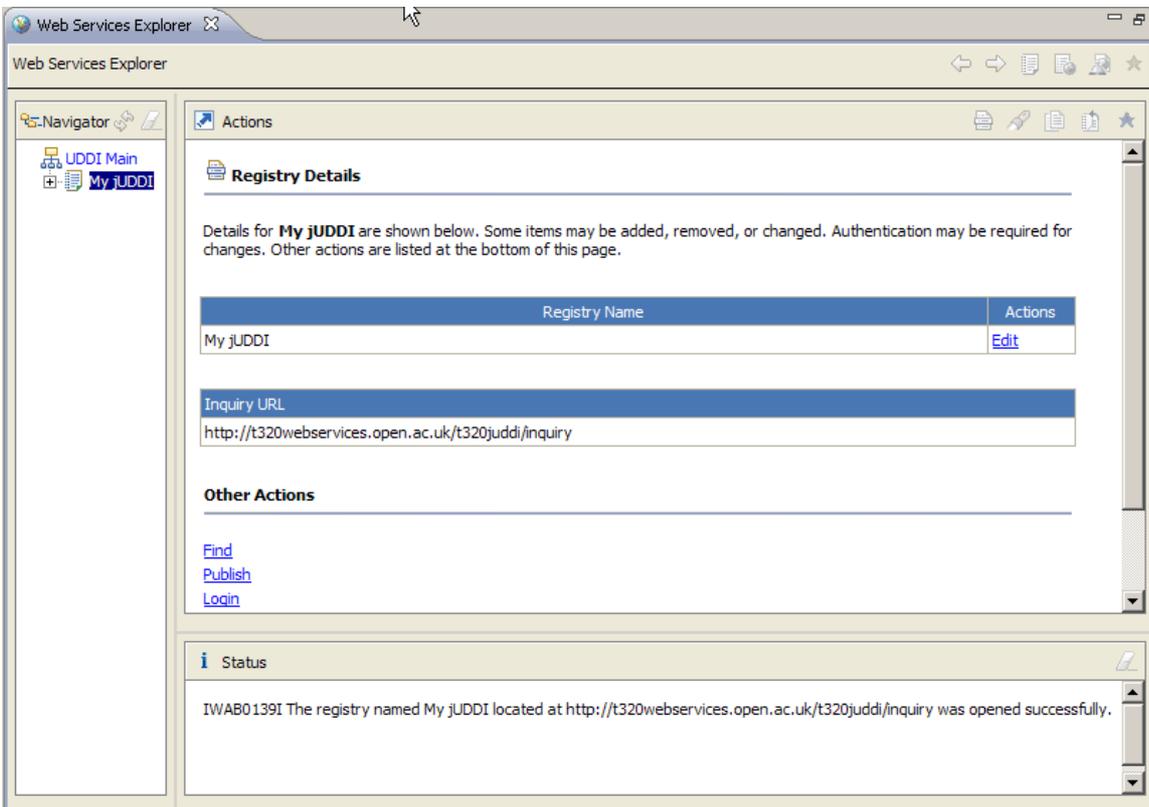
`http://t320webservices.open.ac.uk/t320juddi/inquiry`

Enter this URL for the UDDI into the 'Inquiry URL' field of the 'Open Registry' page and give the registry a name (Figure 4). Then click on the 'Go' button.



**Figure 4** OU UDDI URL entered into 'Open Registry' page

After a time the UDDI should open and you should see your name for the UDDI listed in the Navigator panel. In the Status panel at the bottom, a message stating that the registry was opened successfully should be listed (Figure 5).



**Figure 5** Web Services Explorer after opening UDDI

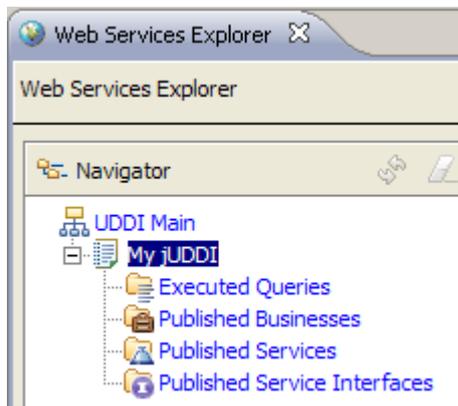
The 'Registry Details' page is typical of other pages you will see in the Explorer. There is an additional set of five buttons at the top right-hand corner of this page. These also provide rapid access to common operations that you might want to perform, given that you have opened the UDDI. Again, hovering over a button will display a tool-tip description of the button.

At the top of the page is a set of data items that relate to the subject of the page. These consist of a 'Registry Name' and 'Inquiry URL'.

At the bottom of the page is a set of 'Other Actions' links. These vary according to the page displayed and provide access to typical actions you may want to take, given the current context.

In the 'Registry Details' list of 'Other Actions' you will find an option called 'Add to Favorites'. If you select this option then the jUDDI ('My jUDDI') will be added to the predefined pull-down list of registries under 'Favorite Registry' on the 'Open Registry' page that you saw in Figure 4. This means that if you return to the registry after closing Eclipse, you need only select the 'My jUDDI' option to fill the 'Inquiry URL' and 'Registry Name' fields.

The UDDI in the Navigator on the left can be expanded to reveal the children under the registry (Figure 6). Later, as you perform further actions, actual instances of items will be added under the four initial groupings.



**Figure 6** Groups of items under UDDI in Navigator

The first item, 'Executed Queries', is used to hold the results of queries, such as searches for businesses or services, so that they can be referred to later. The query results are listed but are held only within Eclipse and not within the UDDI. The other three items group together the instances of the descriptions that have been published under three folders: business descriptions, service descriptions and service interfaces.

To publish in the jUDDI you must have a valid login and password. If you have set up your own jUDDI installation then when you ran the SQL script to create database tables you will have seen that users are created in the 'PUBLISHER' table. The SQL which does this is in the insert\_publishers.sql file which has a statement similar to:

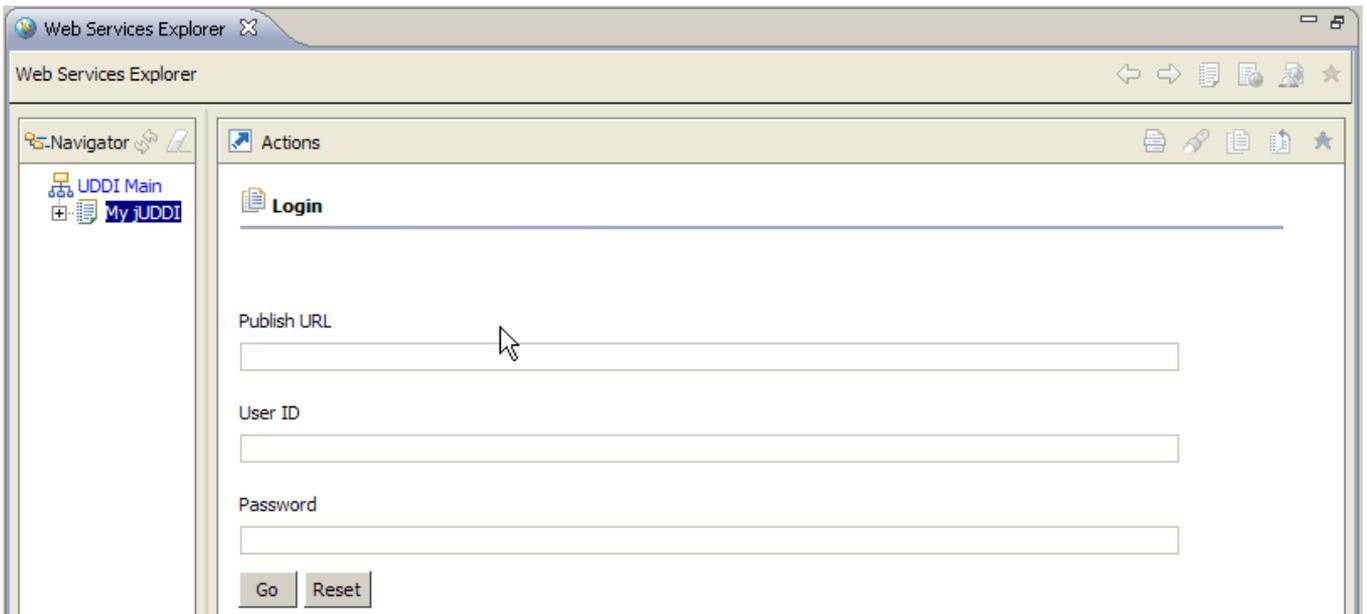
```
INSERT INTO PUBLISHER
(PUBLISHER_ID,PUBLISHER_NAME,EMAIL_ADDRESS,IS_ENABLED,IS_ADMIN)
VALUES ('jdoe','John Doe','john.doe@apache.org','true','true');
```

To publish in the jUDDI, you need first to log in. At the bottom of the Actions panel there is a list of 'Other Actions' that includes a 'Login' link. Click on this link now. A 'Login' page will then appear in the Actions panel (Figure 7).

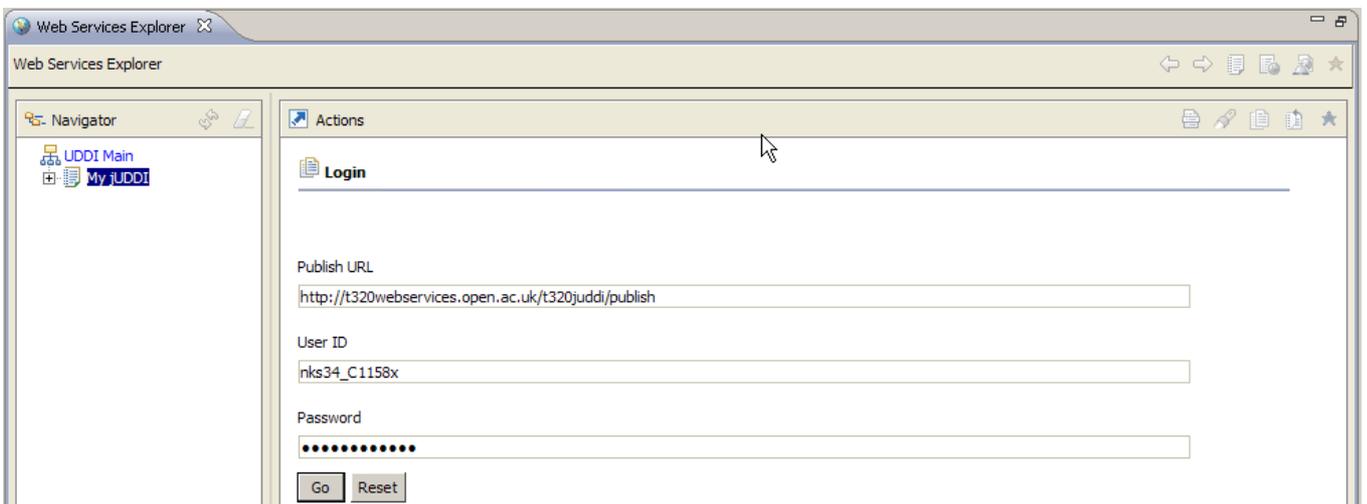
To log in you should type the URL that will be used for publishing items by Eclipse, your User ID and a password.

The 'Publish URL' for the OU jUDDI is:

```
http://t320webservices.open.ac.uk/t320juddi/publish
```



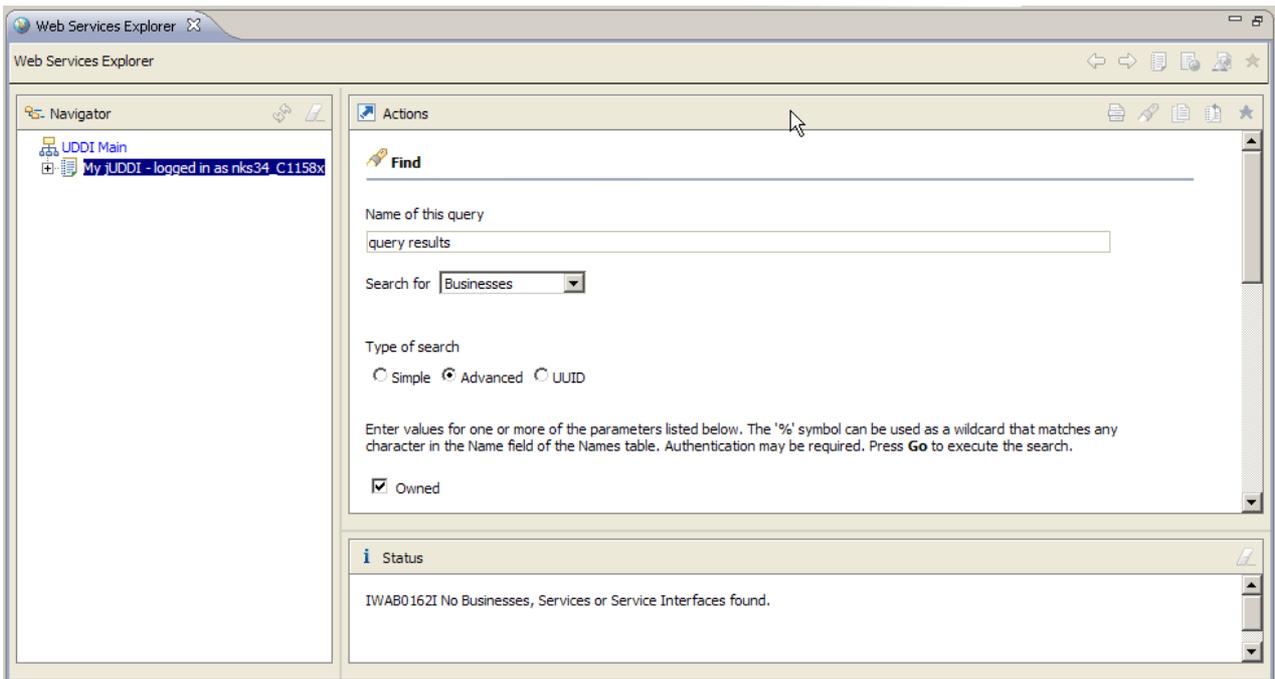
**Figure 7** Web Services Explorer UDDI 'Login' page



**Figure 8** Completed log-in credentials for user 'nks34\_C1158x'

A password is not actually required for you're the jUDDI account as these are not actually implemented for login, so keep your User ID secret, but you can type in your User ID again (Figure 8).

After a time the log-in process will complete. You will see that you are logged in and that the jUDDI listed in the Navigator shows this (9). You will also see that a new 'Find' page is presented in the Actions panel and that a Status message is reporting that no businesses, services or interfaces have been found.



**Figure 9** Web Services Explorer after initial log-in

Now that you are logged in, you can start to publish entries. Note that there is a possibility that you may become logged out, in which case you should see a message in the Status panel to this effect and should then log in again.

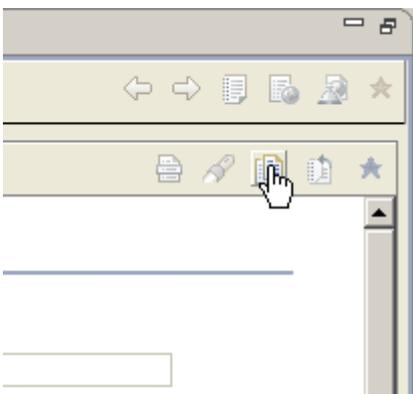
## Publishing entries

### Publishing a business description

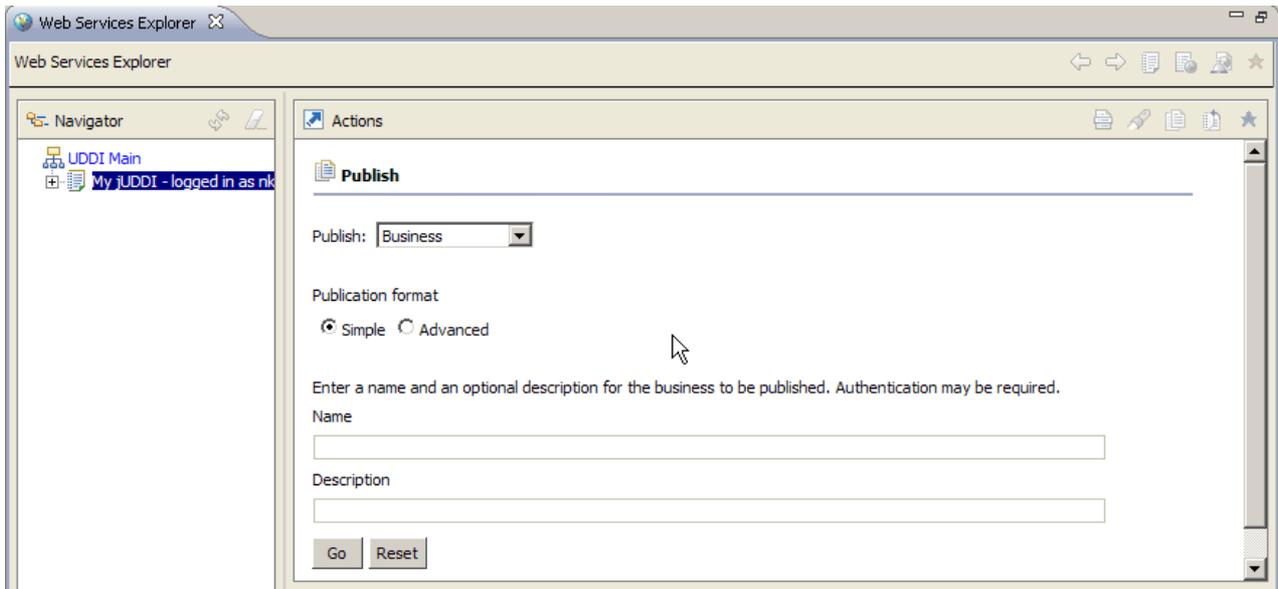
You will start by publishing a business description to the registry. You can only publish a service if a business description exists that it can be associated with.

First switch to the 'Publish' action view by clicking on the 'Publish' shortcut, as shown in Figure 10 (you might also like to hover the mouse over the other shortcuts to determine their purpose). You will then see the 'Publish' page displayed in the Actions panel (Figure 81).

Enter a name and description for your business.

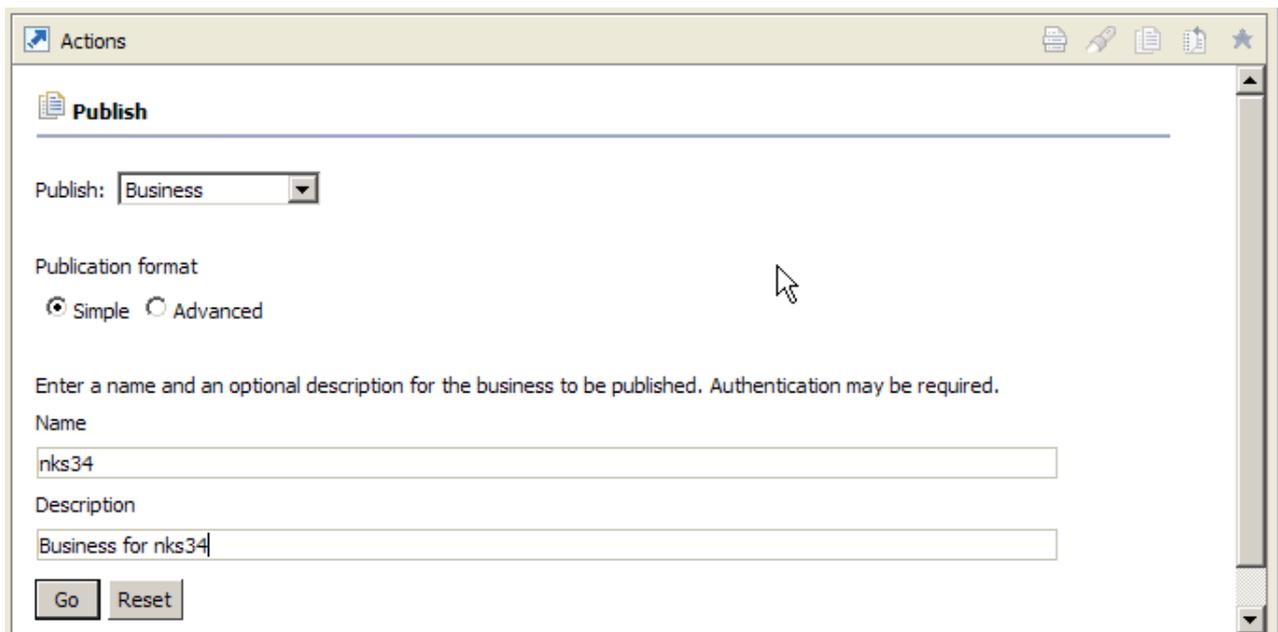


**Figure 10** Shortcut button to 'Publish' page



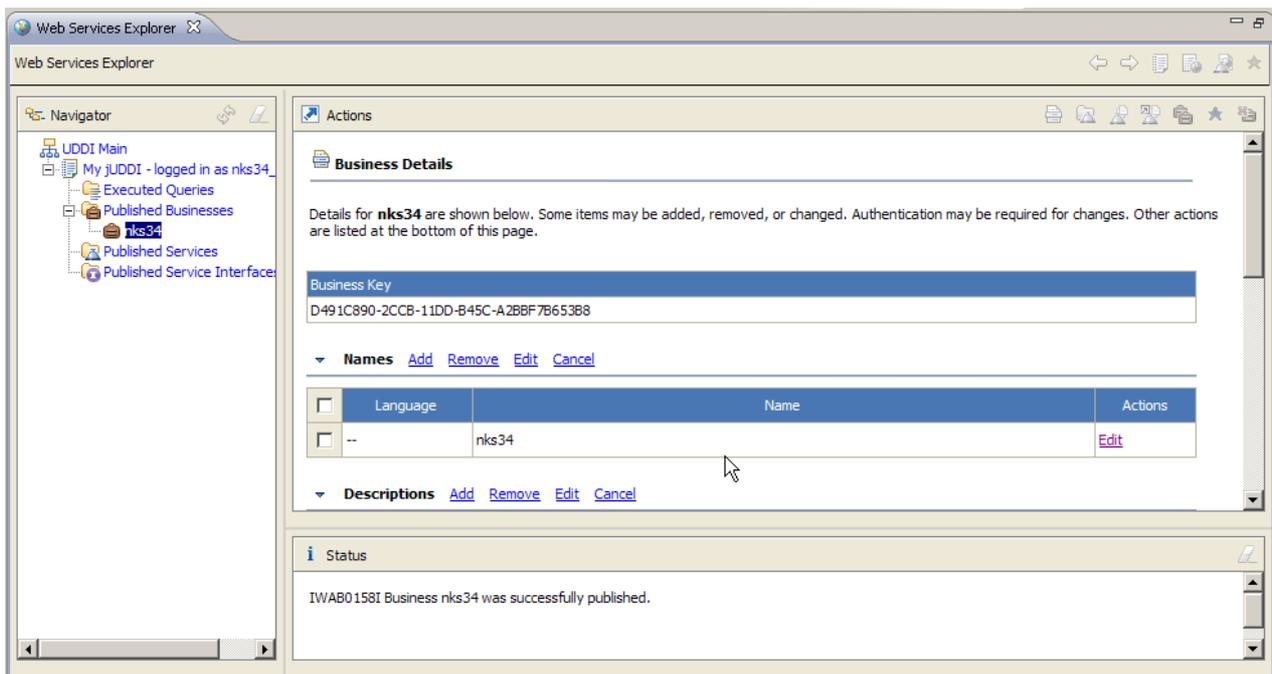
**Figure 8** 'Publish' page in Actions panel

Once you have entered your business details (Figure 92), click on the 'Go' button to publish your business description.



**Figure 92** Completed business details for publishing

You should then see a number of changes (Figure 103).



**Figure 103** Web Services Explorer after publishing business

The Actions panel will list 'Business Details' for the newly entered business and in the Status panel you should see a message reporting that the business was successfully published.

The business, named after your OUCU, will be listed under 'Published Businesses' in the Navigator. If in the future you want to return to the 'Business Details' page for this particular business, you can click on 'Published Businesses' in the Navigator and then click on the link to the business you want in the list of businesses.

If you examine the 'Business Details' page, you will see that it consists of a range of information under different headings. You have entered a very minimal set of information, which is listed under the headings that have information displayed underneath. These are (you may need to scroll down to see them all):

- the 'Business Key', which the registry has generated for you as a unique identifier for your business
- 'Names', where the single name for the business (here 'nks34') is listed
- 'Descriptions', where the short description of the business you gave earlier is listed
- 'Discovery URLs', which lists a single URL that on first sight seems incorrect. However, if you know that 't320webservices.open.ac.uk' is an alias for 'dotterel.open.ac.uk' then this URL may be more easily understood. You will use this URL shortly.

Other headings have no associated information and are thus not expanded in the initial view of the business's information. You can show or hide the information under a heading by clicking on the small arrow next to it.

The 'Discovery URL' given for my example business entry is:

```
http://dotterel.open.ac.uk/t320juddi/uddiget.jsp?businesskey=
D491C890-2CCB-11DD-B45C-A2BBF7B653B8
```

This points to the UDDI on the server I have just published to and then requests a Java Server Page (JSP) called 'uddiget.jsp', passing over the `businesskey` value to the page. The JSP provides a set of basic functionality so that given a key value (`businesskey`, `servicekey`, `bindingkey` or `tmodelkey`) it will return an XML document describing the entry identified by the key.

So, if you type your own 'Discovery URL' into a browser then you can check that your entry has been published. Figure 114 demonstrates the result using the example

above. (Please note that some problems have been identified with displaying the XML in some versions of the Firefox browser.)

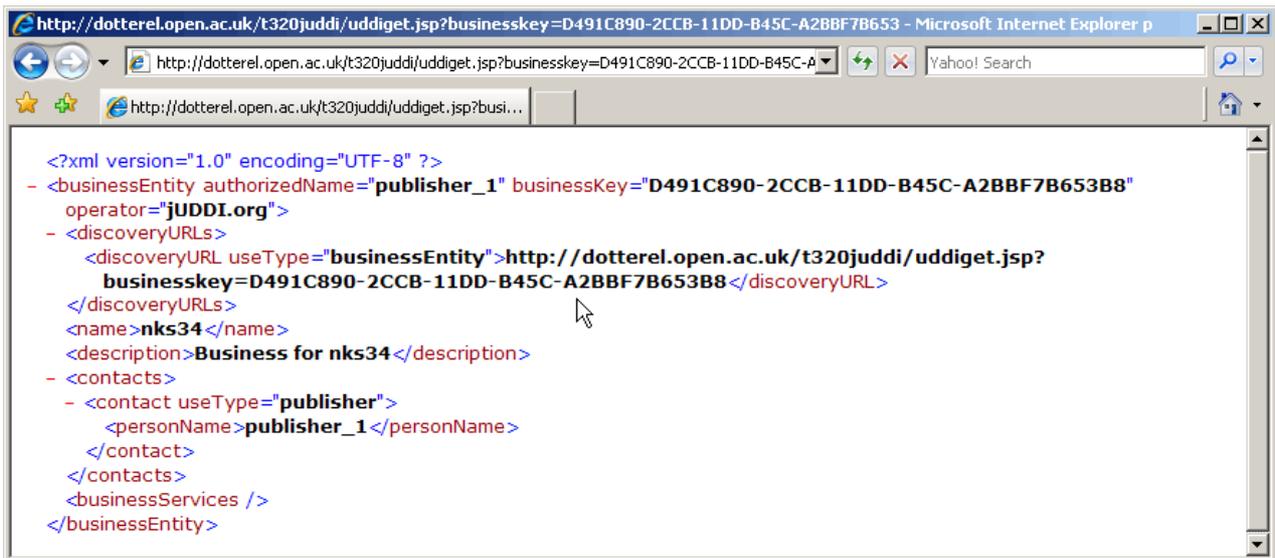


Figure 114 businessEntity description retrieved from Juddi

## Publishing services

Having published a business, you can now move on to publishing one or more web service descriptions that are associated with the business description.

At the bottom of the 'Business Details' page is a list of 'Other Actions' you can take, which are generally useful after publishing a business description. Select the 'Publish Service' link so that the page in Figure 5 is shown.

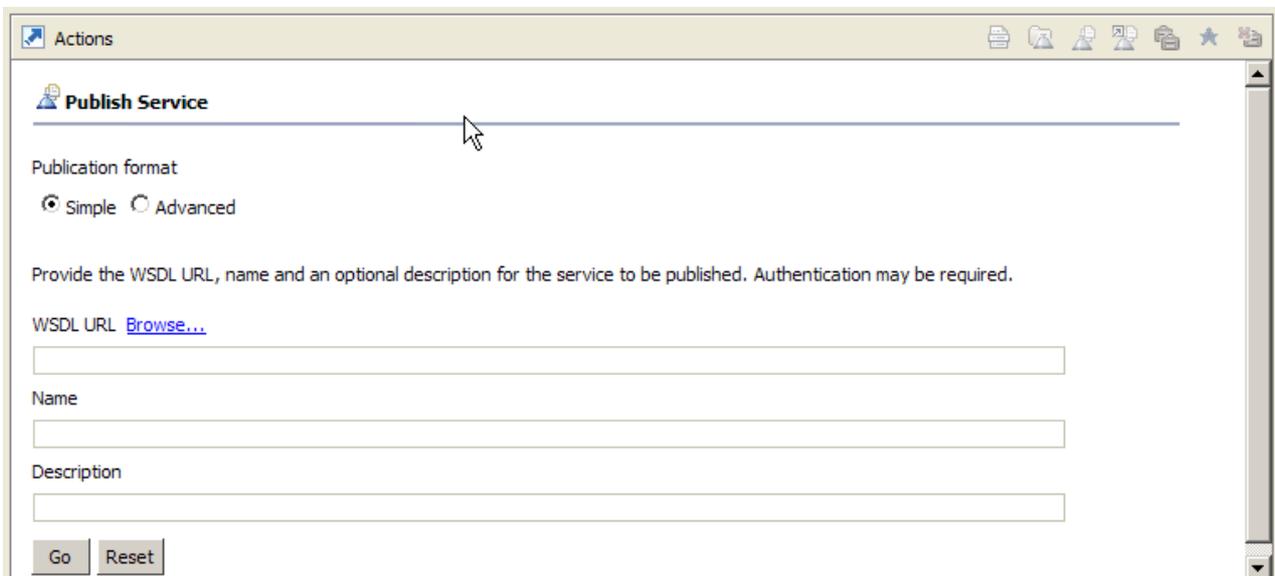


Figure 15 'Publish Service' page

Now you need to refer back to your earlier deployment of the 'Hello' web service. The essential item you require is the URL to obtain the service's WSDL description.

When we deployed the service we saw that the WSDL was at:

```
http://t320webservicess.open.ac.uk/axis_nks34_axis2_A11581/
services/HelloService?wsdl
```

So enter **your own** service's URL and complete the 'Name' and 'Description' fields (Figure 126). Notice that I have used a personal identifier as the initial part of the service name so that I can easily distinguish it from other services. This is generally good practice in a public UDDI.

**Actions**

**Publish Service**

Publication format  
 Simple  Advanced

Provide the WSDL URL, name and an optional description for the service to be published. Authentication may be required.

WSDL URL [Browse...](#)

Name

Description

**Figure 126** Completed 'Publish Service' fields

Then click the 'Go' button. After a short time you should see the Web Services Explorer change to display similar information to that shown in Figure 137.

**Java - http://localhost:65141/wsexplorer/wsexplorer.jsp?org.eclipse.wst.ws.explorer=0 - Eclipse Platform**

File Edit Navigate Search Project Run Window Help

**Web Services Explorer**

Web Services Explorer

**Navigator**

- UDDI Main
  - My jUDDI - logged in as nks34\_C1
    - Executed Queries
    - Published Businesses
    - nks34
      - Published Services
        - nks34HelloService**
        - Published Service Interfaces
        - http://t320.open.ac.uk

**Actions**

**Service Details**

Details for **nks34HelloService** are shown below. Some items may be added, removed, or changed. Authentication may be required for changes. Other actions are listed at the bottom of this page.

Service Key

WSDL URL	Actions
http://t320webservices.open.ac.uk:80/axise_nks34_axis2_A11581/services/HelloService?wsdl	<a href="#">Edit</a>

**Names** [Add](#) [Remove](#) [Edit](#) [Cancel](#)

Language	Name	Actions
<input type="checkbox"/>	nks34HelloService	<a href="#">Edit</a>

**Status**

IWAB0160I Service interface http://t320.open.ac.uk was successfully published.  
 IWAB0159I Service nks34HelloService was successfully published.

**Figure 137** Web Services Explorer after publishing 'Hello' service

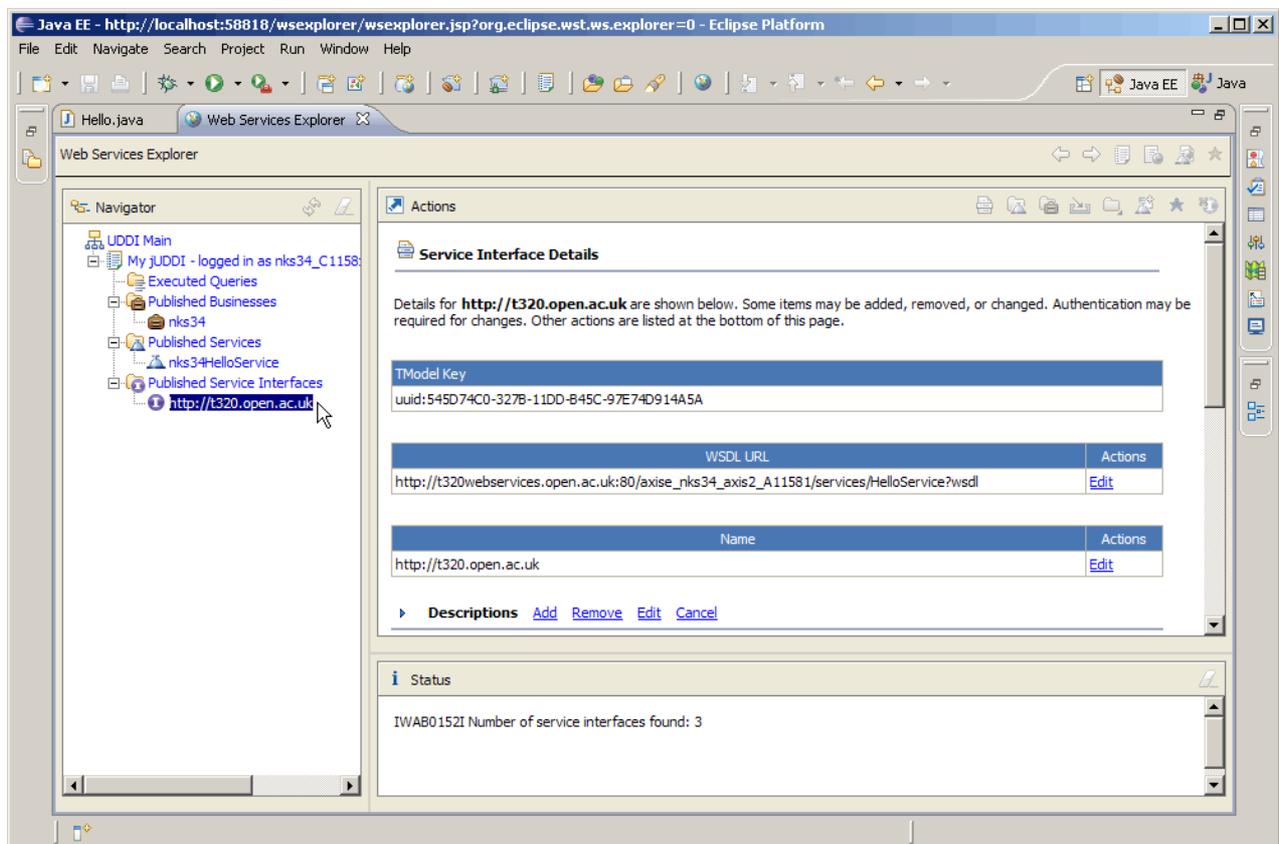
The information displayed is much the same as for the earlier business (with a key etc.) but, of course, relates to a service.

If you examine the Status pane you will see that as well as a service, a 'Service interface' was also published.

What you may not have realised is that Eclipse has actually accessed the service WSDL to gather the information to create the service interface. If your service is not deployed to access and the WSDL is not available then you will see an error message.

The information from the service's WSDL, together with the WSDL URL, is used to create a service interface. To view the service interface you can click on it under 'Published Service Interfaces' (it will be of the form 'http://t320.open.ac.uk') and then, if necessary (depending on your previous actions), click on the 'Details' button (the leftmost button in the set in the corner of the Actions pane).

The details of the service interface are shown in Figure 148.



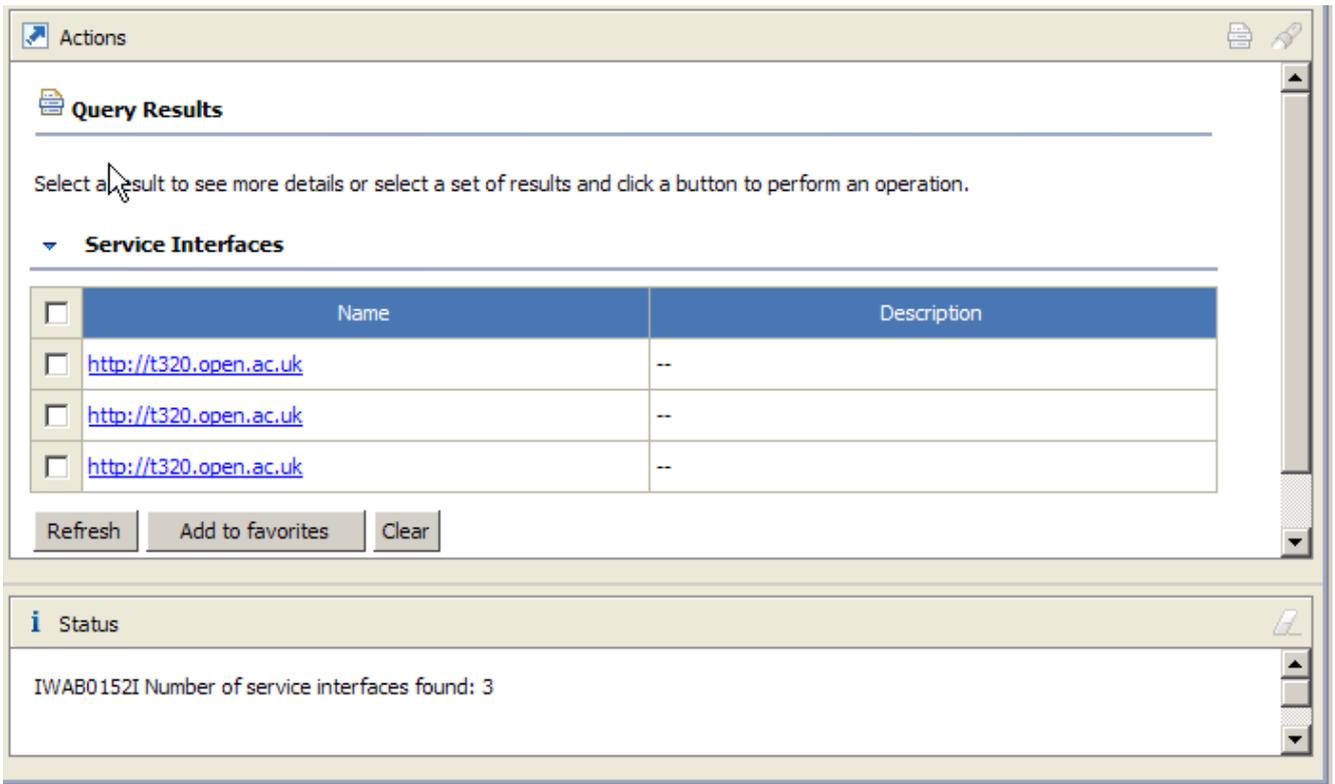
**Figure 148** 'Service Interface Details' page

Alternatively you can issue a search query for interfaces. To do this you can, for example, click on the service in the Navigator (and then on the 'Details' button if necessary) to view the 'Service Details' page. Then, from the 'Other Actions' list at the bottom of the details, select 'Get Service Interfaces'. A 'Query Results' page will be displayed listing the service interfaces found (Figure 191519).

You will notice that in this case, the service interface is actually listed three times. Presumably this behaviour is due to the same interface being found by listing items of type 'interface', by listing service interfaces and by listing business interfaces. Click on any one of the listed items to display the same details (Figure 148).

On the 'Service Interface Details' page, notice that the service interface lists a tModel key value. So we know that a tModel has been created in the jUDDI database. In fact it seems that the Eclipse web service explorer simply terms tModels as 'Service Interfaces' which simply include the information we can expect in a tModel. Later in

this guide you will learn how to use another tool, the 'jUDDI Console' to view UDDI entries. You can then take the Service Interface's tModel key and use this to view the actual tModel itself.



**Figure 1915** Result of search for service interfaces

Before moving on notice that the list of 'Other Actions' at the bottom of the main panel now contains typical related actions and other actions specific to the service interface such as 'Unpublish Service Interface' (Figure 20).

### Other Actions

[Get Services](#)  
[Get Businesses](#)  
[Import WSDL To workbench](#)  
[Import WSDL To File System](#)  
[Launch Web Service Wizard](#)  
[Add To Favorites](#)  
[Unpublish Service Interface](#)

**Figure 20** Other Actions list for Service Interface

This action will delete the UDDI entry for the Service Interface, after taking you on to another screen where you confirm the action.

You will find that in general when a specific entry is displayed, such as a business or service then the 'Other Actions' will include actions specific to that entry type, such as 'Unpublish Business' when a Business entry is displayed. These actions allow you to delete any entries that you have created in error.

Before I move on to using jUDDI to access services, which is very simple using Eclipse, I shall briefly revisit the jUDDI Console that you saw earlier when I looked at the OU server accounts.

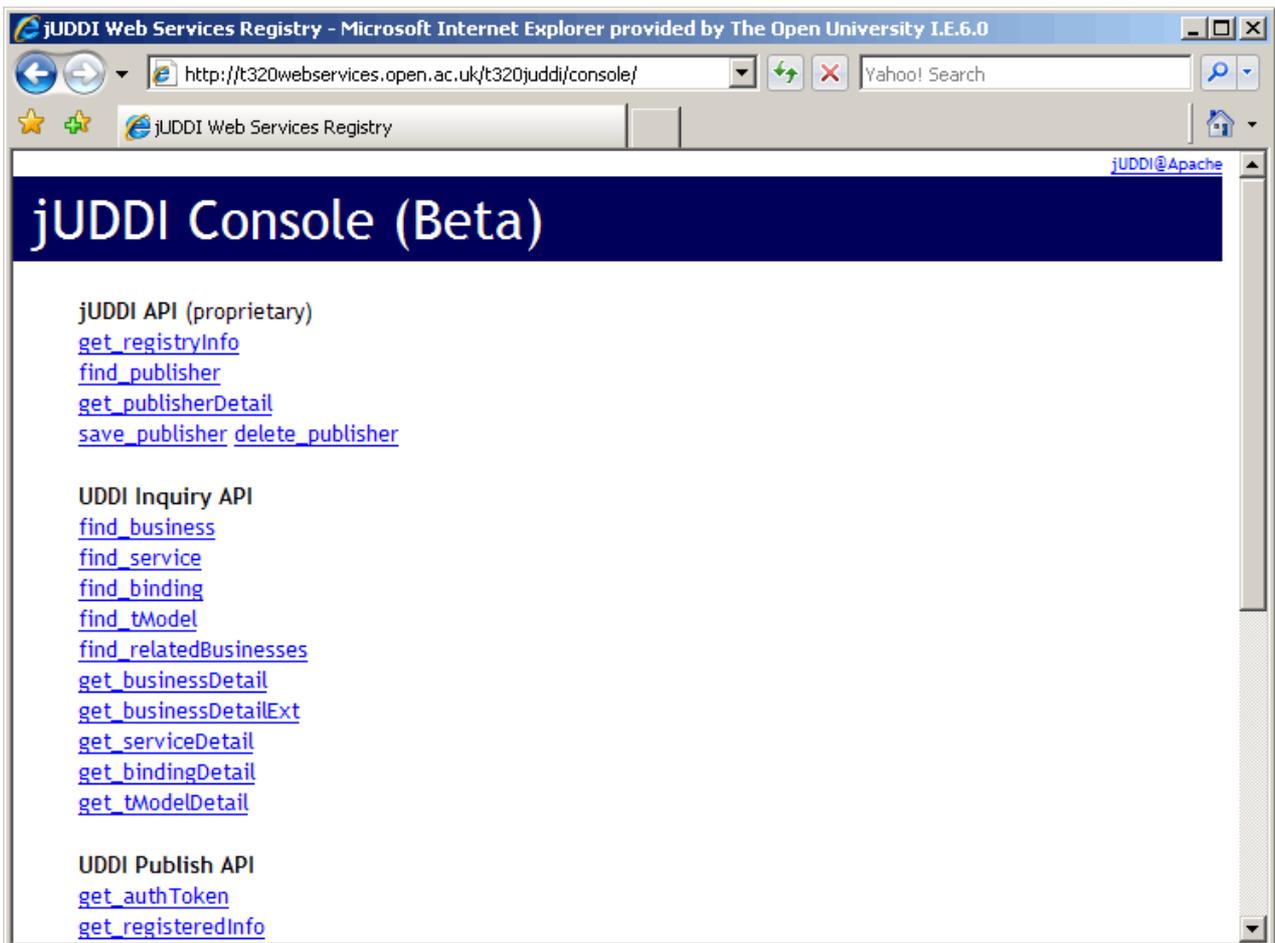
## Publishing with the jUDDI Console

### The Console

The jUDDI Console provides an alternative means of publishing entries to jUDDI and searching for entries. The view it provides is closer to the actual UDDI data structures that you should expect than the Eclipse Web Services Explorer you have just been using. The Console is, however, slightly less 'user friendly' in certain ways.

The Console is found under the top-level URL of your jUDDI installation with the postfix '/console' (Figure 20).

You can search the jUDDI using the Console without providing any identity information, but to publish entries you need to log in and acquire an 'authToken'. An authToken is issued to you by jUDDI based on your log-in and password, and is then used in any action that requires appropriate permission, such as publishing an entry. Each authToken has a limited lifetime. If your token expires, you will see a message to this effect in the Status screen. You can then use your log-in and password to acquire a new token.



**Figure 160** jUDDI Console

As you saw earlier, the Console groups actions under three headings: 'jUDDI API (proprietary)', 'UDDI Inquiry API' and 'UDDI Publish API'. In general, the 'Publish API' actions require that you provide an authToken whereas the other actions do not.

You can only interact with jUDDI via the Console using XML structures. So, for example, if you want to search for a business, you can first ask for the 'find business' XML structure template (i.e. a 'skeleton' document) into which you then enter any information you have, such as the name of the business, before you submit the XML document that is used in searching.

Within any of these 'action templates' there are XML elements that are mandatory (for example, in publishing any entry you must provide an authToken value) and those that are optional (for example, in searching for a service you may enter a tModel UUID or choose not to).

Note: if you wish actually to publish a business and web service using the Console and you have already done so using Eclipse, you should return to Eclipse and delete your business, service and service interface entries before proceeding. Alternatively you can use the Console to search for the entries that you have created so far.

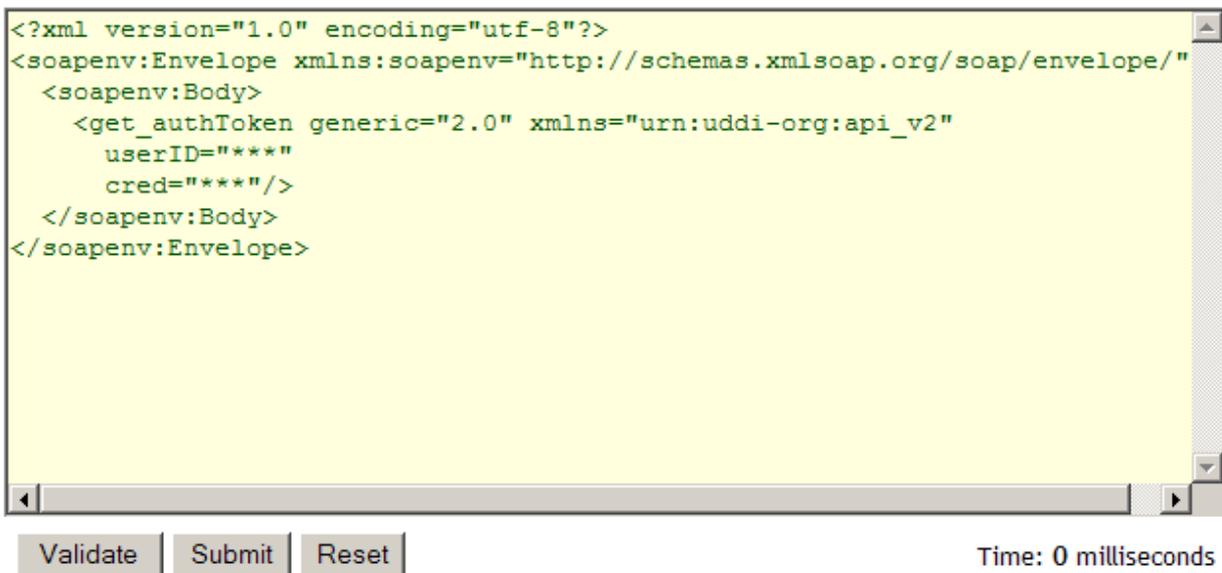
## Authenticating

To acquire an authToken, first click on the 'get\_authToken' link under the 'UDDI Publish API' heading. This will present you with the XML template shown in Figure 171.

# jUDDI Console (Beta)

## get\_authToken

The [get\\_authToken](#) API call is used to obtain an authentication token ([authToken](#)). Authentication tokens are opaque values that are required for all other publisher API calls. If an error occurs while processing this API call, a [dispositionReport](#) element will be returned to the caller within a [SOAP Fault](#) containing information about the [error](#) that was encountered.



```
<?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="****"
      cred="****"/>
    </soapenv:Body>
  </soapenv:Envelope>
```

Validate Submit Reset

Time: 0 milliseconds

**Figure 171** Acquiring an authToken

At this point, notice that below the 'template XML' field are three buttons and then another blank field. This illustrates how the Console operates in general. Selecting a link from one of the API lists presents you with an XML template in the upper field.

You are then expected to fill in appropriate parts of the template before clicking on the 'Submit' button, which sends the XML to jUDDI. After a short time jUDDI will respond with an appropriate XML document, which is displayed in the lower field below the buttons.

The other two buttons are as follows:

- 'Reset', which returns the XML template to its original 'blank' state, thus removing any editing you have carried out
- 'Validate', which is intended to check the XML syntax of a template you have edited, but this functionality is not yet implemented.

The XML template for 'get\_authToken' consists of a SOAP envelope around a SOAP body and the actual <get\_authToken> element. Inside this element are two items, userID and cred. Each of these has a value given as " \* \* \* ". This again illustrates a general principle of the Console: values that appear as '\*\*\*\*' in the template are intended to be replaced by actual values.

So now replace the three stars after 'userID' with your jUDDI username. As I mentioned earlier, jUDDI doesn't currently use a password, here called 'cred', so the value of 'cred' can be left as three stars.

Next, click the 'Submit' button. After a time you will see that a response is displayed in the lower text box of the Console. The XML structure will be similar to the following:

```
<?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:A44EFF50-340E-11DD-BF50-AF1574038CC8</authInfo>
    </authToken>
  </soapenv:Body>
</soapenv:Envelope>
```

The important element in the response is the <authInfo> element, which carries the token that you will use later to publish items in jUDDI. Here the value is:

```
authToken:A44EFF50-340E-11DD-BF50-AF1574038CC8
```

You might like to copy your authToken value to, say, Notepad so that it is available for use later.

If your authToken expires then the action you are performing, if it requires an authToken, will fail. In this case you will see a response that includes an error message:

```
<?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>
    <soapenv:Fault>
      <faultcode>soapenv:Client</faultcode>
      <faultstring>E_authTokenExpired (10110) Authentication
token information has timed out. authToken:
authToken:A44EFF50-340E-11DD-BF50-AF1574038CC8</faultstring>
      <detail>
        <dispositionReport generic="2.0" operator="jUDDI.org"
xmlns="urn:uddi-org:api_v2">
          <result errno="10110">
            <errInfo errCode="E_authTokenExpired"
xsi:type="xsd:string">E_authTokenExpired (10110) Authentication
token information has timed out. authToken:
```

```

    authToken:A44EFF50-340E-11DD-BF50-AF1574038CC8</errInfo>
  </result>
</dispositionReport>
</detail>
</soapenv:Fault>
</soapenv:Body>
</soapenv:Envelope>

```

You will then need to use 'get\_authToken' again to get a new token.

## Publishing

Again, you will first publish a business and then publish service entries for the services that the business provides.

Click on the 'save\_business' link in the 'Publish API' list. This will display the following XML template:

```

<?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>***</authInfo>
      <businessEntity businessKey="">
        <name>***</name>
        <description>***</description>
        <contacts>
          <contact useType="***">
            <personName>***</personName>
            <phone>***</phone>
            <email>***</email>
          </contact>
        </contacts>
      </businessEntity>
    </save_business>
  </soapenv:Body>
</soapenv:Envelope>

```

This listing contains a range of items that have three stars as a value. These are the items that can be 'filled in'. In fact, some of these items must be filled in, although the rules for this are not clear. It is sensible that we must fill in at least the <authInfo> and one element of the business description, such as the business <name> element. This would provide our identity, which may be used to check that we have permission to publish, and a minimal amount of information describing the business.

So now fill in the template by adding your authToken and filling in the 'name', 'description' and 'contact' information in the template. A completed template is given below. Note that the 'useType' for the contact can be any reasonable value, such as 'localOffice'.

```

<?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:A44EFF50-340E-11DD-BF50-AF1574038CC8</authInfo>
      <businessEntity businessKey="">
        <name>nks34Business</name>
        <description>The business of nks34</description>
        <contacts>
          <contact useType="headOffice">
            <personName>nks</personName>
            <phone>01908881517</phone>
          </contact>
        </contacts>
      </businessEntity>
    </save_business>
  </soapenv:Body>
</soapenv:Envelope>

```

```

        <email>nks34@open.ac.uk</email>
    </contact>
</contacts>
</businessEntity>
</save_business>
</soapenv:Body>
</soapenv:Envelope>

```

In response to submitting the template you will receive an XML response in the lower window, similar to the following:

```

<?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="publisher_1"
businessKey="77A466A0-346F-11DD-A6A0-AEE7C8000C33" operator="jUDDI.org">
        <discoveryURLs>
          <discoveryURL useType="businessEntity">http://dotterel.open.ac.uk/t320juddi/
uddiget.jsp?businesskey=77A466A0-346F-11DD-A6A0-AEE7C8000C33</discoveryURL>
        </discoveryURLs>
        <name>nks34Business</name>
        <description>The business of nks34</description>
        <contacts>
          <contact useType="headOffice">
            <personName>nks</personName>
            <phone>01908881517</phone>
            <email>nks34@open.ac.uk</email>
          </contact>
        </contacts>
      </businessEntity>
    </businessDetail>
  </soapenv:Body>
</soapenv:Envelope>

```

There are three interesting items in the businessDetail returned by jUDDI:

- 'authorizedName', which is actually a name given to you according to your log-in. This is normally assigned by the UDDI administrator when creating your log-in and password. As the log-in and password system for jUDDI is incomplete, you don't need this name and don't need to be concerned with it.
- 'businessKey', which is the unique key generated by jUDDI that identifies this single business entry. You might want to copy this to Notepad for use in other templates.
- 'discoveryURL', which is the URL of the jUDDI itself. This is extended to point to a JSP called 'uddiget.jsp'. This page supports access to UDDI entries over HTTP. The URL also includes the businessKey so that this specific business can be located. You can test this yourself by entering the URL into a browser. The HTTP response will list the same XML as in the listing above.

Now you have a business entry in the registry. We want to complete a set of entries for the web service that will consist of a service description and at least one binding and one tModel (Figure 1).

In Figure 1 I depicted the relationships (one-to-one, one-to-many, etc.) of the different entries in UDDI. So, for example, a business entry can have one or more corresponding service entries. What Figure 1 does not depict is the precise references that are made between the different types of entry. If we publish a tModel then the UDDI will assign this a unique 'tModel Key' value, in a similar fashion to the 'businessKey' assigned to the business.

When we publish a binding, the description we submit to jUDDI needs to include a `serviceKey` value. When we publish a service, the description should include a `tModelKey` and a `businessKey`.

So, as you have a `businessKey`, you next need to publish a `tModel` for the service. Click on the 'save\_tModel' link in the jUDDI Console. You will then be presented with the following XML template:

```
<?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>***</authInfo>
      <tModel tModelKey="">
        <name>***</name>
        <description>***</description>
        <overviewDoc>
          <description>***</description>
          <overviewURL>***</overviewURL>
        </overviewDoc>
        <identifierBag>
          <keyedReference tModelKey="***" keyName="***" keyValue="***" />
        </identifierBag>
        <categoryBag>
          <keyedReference tModelKey="***" keyName="***" keyValue="***" />
        </categoryBag>
      </tModel>
    </save_tModel>
  </soapenv:Body>
</soapenv:Envelope>
```

Enter appropriate information into the elements with '\*\*\*' values as follows:

- for `<authInfo>` enter your `authToken` value
- for `<name>` and `<description>` enter appropriate values, using your own OUCU in the same fashion as shown below
- for `<overviewDoc>` enter a description and the URL for your service's WSDL into the two child elements `<description>` and `<overviewURL>`.

Other fields that have '\*\*\*' values can be left as they are. You should end up with something like the following:

```
<?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:A44EFF50-340E-11DD-BF50-AF1574038CC8</authInfo>
      <tModel tModelKey="">
        <name>nks34TmodelForHelloService</name>
        <description>nks34 tModel entry for the Hello service</description>
        <overviewDoc>
          <description>WSDL for the Hello Service</description>
          <overviewURL>http://t320webservices.open.ac.uk/
            axis_nks34_axis2_A11581/services/HelloService?wsdl</overviewURL>
        </overviewDoc>
        <identifierBag>
          <keyedReference tModelKey="***" keyName="***" keyValue="***" />
        </identifierBag>
        <categoryBag>
          <keyedReference tModelKey="***" keyName="***" keyValue="***" />
        </categoryBag>
      </tModel>
    </save_tModel>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    </save_tModel>
  </soapenv:Body>
</soapenv:Envelope>

```

Then click the 'Submit' button to send the XML to jUDDI. The response to the 'save\_tModel' request should be similar to the following:

```

<?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="publisher_1" operator="jUDDI.org"
tModelKey="uuid:B6CB6500-34B9-11DD-A500-84FAA8E6EC03">
        <name>nks34TmodelForHelloService</name>
        <description>nks34 tModel entry for the Hello service</description>
        <overviewDoc>
          <description>WSDL for the Hello Service</description>
          <overviewURL>http://t320webservices.open.ac.uk/
axis_nks34_axis2_A11581/services/HelloService?wsdl</overviewURL>
        </overviewDoc>
        <identifierBag>
          <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
        </identifierBag>
        <categoryBag>
          <keyedReference keyName="***" keyValue="***" tModelKey="***"/>
        </categoryBag>
      </tModel>
    </tModelDetail>
  </soapenv:Body>
</soapenv:Envelope>

```

Notice that you now have a tModelKey value, so you can progress to publish a service. You might want to copy this to Notepad for use in other templates.

Now click on the 'save\_service' link, which will present you with the following XML template:

```

<?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:uddi-org:api_v2">
      <authInfo>***</authInfo>
      <businessService businessKey="***" serviceKey="">
        <name>***</name>
        <description>***</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="">
            <accessPoint URLType="http">***</accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo tModelKey="***">
                <instanceDetails>
                  <overviewDoc>
                    <overviewURL>***</overviewURL>
                  </overviewDoc>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>
        </bindingTemplates>
      </businessService>
    </save_service>
  </soapenv:Body>
</soapenv:Envelope>

```

```

    </save_service>
  </soapenv:Body>
</soapenv:Envelope>

```

Fill in the 'save\_service' template with the values for 'authInfo', 'businessKey' and 'tModelKey' that you obtained earlier, and then also fill in a name, description, access point (which is the URL for the endpoint, not the WSDL) and overview URL (which does refer to the WSDL).

The completed template, ready for submission, is shown below:

```

<?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:A1FD9E90-34B9-11DD-9E90-E9024A08B8D0</authInfo>
      <businessService businessKey="77A466A0-346F-11DD-A6A0-AEE7C8000C33"
        serviceKey="">
        <name>nks34HelloService</name>
        <description>the Hello service by nks34</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="">
            <accessPoint URLType="http">http://t320webservices.open.ac.uk/
              axise_nks34_axis2_A11581/services/HelloService</accessPoint>
            <tModelInstanceDetails>
              <tModelInstanceInfo tModelKey="uuid:B6CB6500-34B9-11DD-A500-
                84FAA8E6EC03">
                <instanceDetails>
                  <overviewDoc>
                    <overviewURL>http://t320webservices.open.ac.uk/
                      axise_nks34_axis2_A11581/services/
                      HelloService?wsdl</overviewURL>
                  </overviewDoc>
                </instanceDetails>
              </tModelInstanceInfo>
            </tModelInstanceDetails>
          </bindingTemplate>
        </bindingTemplates>
      </businessService>
    </save_service>
  </soapenv:Body>
</soapenv:Envelope>

```

Then submit the XML to publish your service description. The result returned will be similar to the following:

```

<?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="77A466A0-346F-11DD-A6A0-AEE7C8000C33"
        serviceKey="6741BEB0-34C0-11DD-BEB0-D431D06AC14C">
        <name>nks34HelloService</name>
        <description>the Hello service by nks34</description>
        <bindingTemplates>
          <bindingTemplate bindingKey="6742D020-34C0-11DD-9020-CDEC2268740C"
            serviceKey="6741BEB0-34C0-11DD-BEB0-D431D06AC14C">
            <accessPoint URLType="http">http://t320webservices.open.ac.uk/
              axise_nks34_axis2_A11581/services/HelloService</accessPoint>
          <tModelInstanceDetails>

```

```

<tModelInstanceInfo tModelKey="uuid:B6CB6500-34B9-11DD-A500-84FAA8E6EC03">
  <instanceDetails>
    <overviewDoc>
      <overviewURL>http://t320webservices.open.ac.uk/
        axise_nks34_axis2_A11581/services/HelloService?wsdl</overviewURL>
    </overviewDoc>
  </instanceDetails>
</tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
</bindingTemplates>
</businessService>
</serviceDetail>
</soapenv:Body>
</soapenv:Envelope>

```

The important item of note is the serviceKey, which informs us that jUDDI has publishes the description. Copy this value to Notepad, as you will need it later.

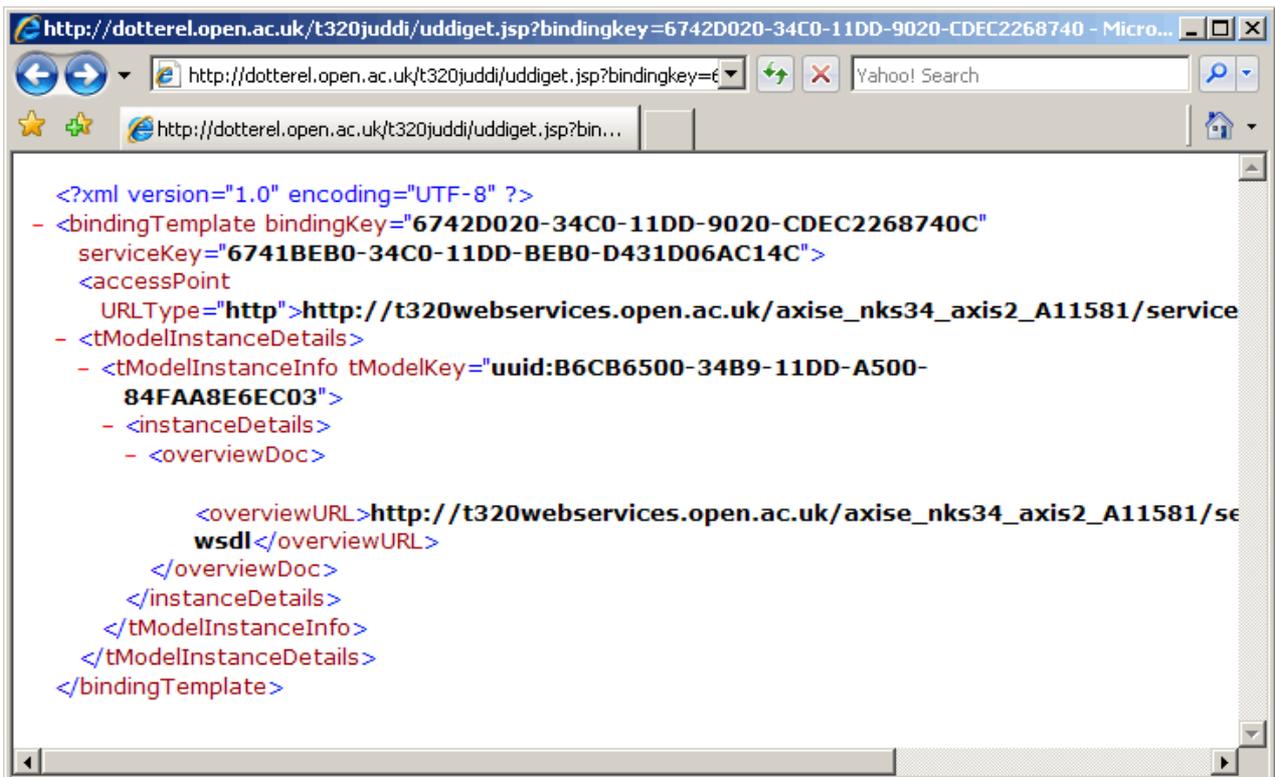
Publishing the service also generates a binding, just as earlier in the Eclipse Web Services Explorer a 'service interface' was generated. The bindingKey is given in the serviceDetail returned by jUDDI (see the listing above). If you copy this key, you can use it to view the binding using the 'uddiget.jsp' page. So in this case, given the bindingKey value, the URL required to get the service binding is:

```

http://dotterel.open.ac.uk/t320juddi/uddiget.jsp?bindingkey=6
742D020-34C0-11DD-9020-CDEC2268740C

```

The XML returned by this URL is shown in Figure 182.



**Figure 182** Service binding displayed in browser

You have now published the four basic descriptions required for your web service.

You may or may not prefer the Console over Eclipse for publishing UDDI entries. It does bring you closer to the underlying UDDI data structures and their content. I have skipped over some of the content that can be added to entries, such as 'keyedReference' items in tModels. If you are interested, you can look up these details in the standards

(see [http://www.uddi.org/taxonomies/Core\\_Taxonomy\\_OverviewDoc.htm](http://www.uddi.org/taxonomies/Core_Taxonomy_OverviewDoc.htm), for example).

Before I leave the Console I shall take a quick look at its search facilities.

## Finding UDDI entries

In the Console, under the 'UDDI Inquiry API' heading, is a list of 'find' and 'get' links. These are intended for retrieving entries from the registry.

To find the binding we have just been looking at, the 'find\_binding' link can be used. This will display the template shown below. There are various items with '\*\*\*' values that can be completed. It's important to appreciate that whilst we left such values in the template, even if we didn't use them, when publishing you should either replace '\*\*\*' with a value or remove the element that encloses the '\*\*\*'.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <find_binding serviceKey="***" maxRows="100" generic="2.0"
      xmlns="urn:uddi-org:api_v2">
      <findQualifiers>
        <findQualifier>***</findQualifier>
      </findQualifiers>
      <tModelBag>
        <tModelKey>***</tModelKey>
      </tModelBag>
    </find_binding>
  </soapenv:Body>
</soapenv:Envelope>
```

Complete the template by inserting your serviceKey. Then remove the <findQualifiers> and <tModelBag> elements so that no values have to be entered in these. You now have a template that resembles the following:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <find_binding serviceKey="6741BEB0-34C0-11DD-BEB0-D431D06AC14C"
      maxRows="100" generic="2.0" xmlns="urn:uddi-org:api_v2">
    </find_binding>
  </soapenv:Body>
</soapenv:Envelope>
```

Submitting this template will retrieve the binding:

```
<?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="6742D020-34C0-11DD-9020-CDEC2268740C"
        serviceKey="6741BEB0-34C0-11DD-BEB0-D431D06AC14C">
        <accessPoint URLType="http">http://t320webservices.open.ac.uk/
          axis_nks34_axis2_A11581/services/HelloService</accessPoint>
        <tModelInstanceDetails>
          <tModelInstanceInfo tModelKey="uuid:B6CB6500-34B9-11DD-A500-84FAA8E6EC03">
            <instanceDetails>
              <overviewDoc>
                <overviewURL>http://t320webservices.open.ac.uk/
                  axis_nks34_axis2_A11581/services/HelloService?wsdl</overviewURL>
              </overviewDoc>
            </instanceDetails>
          </tModelInstanceInfo>
        </tModelInstanceDetails>
      </bindingTemplate>
    </bindingDetail>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    </instanceDetails>
  </tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
</bindingDetail>
</soapenv:Body>
</soapenv:Envelope>

```

### Listing A

Many fields that you fill in within an XML search template, such as keys, need to be given precisely. So it's usual that either you have kept a record of these or they are made available to you from some other search. Other fields, such as 'name', can typically be abbreviated to the stem of the full name.

So, for example, searching for services using the 'find\_service' link, the XML template could be submitted as shown below. This template has a service name that I have specified as 'nks34'. This is the stem of the service name I published earlier ('nks34HelloService'), but as I have published the same service description twice (which is not recommended) there are two service entries that match this stem.

```

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <find_service businessKey="" generic="2.0" xmlns="urn:uddi-org:api_v2">
      <name>nks34</name>
    </find_service>
  </soapenv:Body>
</soapenv:Envelope>

```

Submitting this 'find\_service' template results in the response given below. This contains two serviceInfo items that have the same name but differing keys.

```

<?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>
    <serviceList generic="2.0" operator="jUDDI.org" xmlns="urn:uddi-org:api_v2">
      <serviceInfos>
        <serviceInfo businessKey="77A466A0-346F-11DD-A6A0-AEE7C8000C33"
serviceKey="6741BEB0-34C0-11DD-BEB0-D431D06AC14C">
          <name>nks34HelloService</name>
        </serviceInfo>
        <serviceInfo businessKey="96467290-3273-11DD-B45C-D80431380383"
serviceKey="7B6E5120-3315-11DD-B45C-F714FD6E38BE">
          <name>nks34HelloService</name>
        </serviceInfo>
      </serviceInfos>
    </serviceList>
  </soapenv:Body>
</soapenv:Envelope>

```

The 'get' links, such as 'get\_BindingDetail', retrieve entries based on a specific key value. The 'get\_bindingDetail' XML template is shown below:

```

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <get_bindingDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
      <bindingKey>***</bindingKey>
    </get_bindingDetail>
  </soapenv:Body>

```

```
</soapenv:Envelope>
```

If you insert the `bindingKey` value for your binding and submit the template then you will receive the same response as you saw earlier in Listing A.

The 'delete' links, under the 'Publish API' heading, also function in much the same way as the 'get' links in requiring a specific key for the item to be deleted. The 'delete\_service' template, for example, requires a `serviceKey`:

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

Now I have examined different means of publishing UDDI entries, I shall revisit Eclipse to investigate support in Eclipse for accessing the actual web service being described by UDDI entries.

## Accessing UDDI services

Earlier in this block you saw how to create a simple client in Eclipse to test a web service based on the service's WSDL description. As you have seen now, jUDDI entries can refer to the WSDL document describing a service. It is therefore a simple step to create a client by first accessing a UDDI description and then using the referenced WSDL document to generate a client. Eclipse provides good support for this and another, more direct approach.

### 'Other Actions' list options

If you return to the Web Services Explorer in Eclipse and bring up the 'Details' page for either a Service Interface or a Service then in the 'Other Actions' list you will see some options that are related to WSDL and accessing a service:

- **Import WSDL To workbench.** Allows you to import the WSDL document referenced by the Service or Service Interface into a selected project. After doing so, the WSDL document will appear in the Project Explorer within the selected project and can be used to create a client, as I demonstrated earlier in the block.
- **Import WSDL To File System.** This is much the same as importing to the workbench but you are asked where to save the WSDL document and the WSDL will not appear in the Project Explorer. You might then import the file into Eclipse but this is a longer path to achieving 'Import WSDL To workbench'. Use this option when you want to use or save a WSDL outside Eclipse.
- **Launch Web Service Wizard.** This provides a quick way to generate a client without explicitly downloading WSDL. I shall look quickly at this option now.

### Web Service Wizard

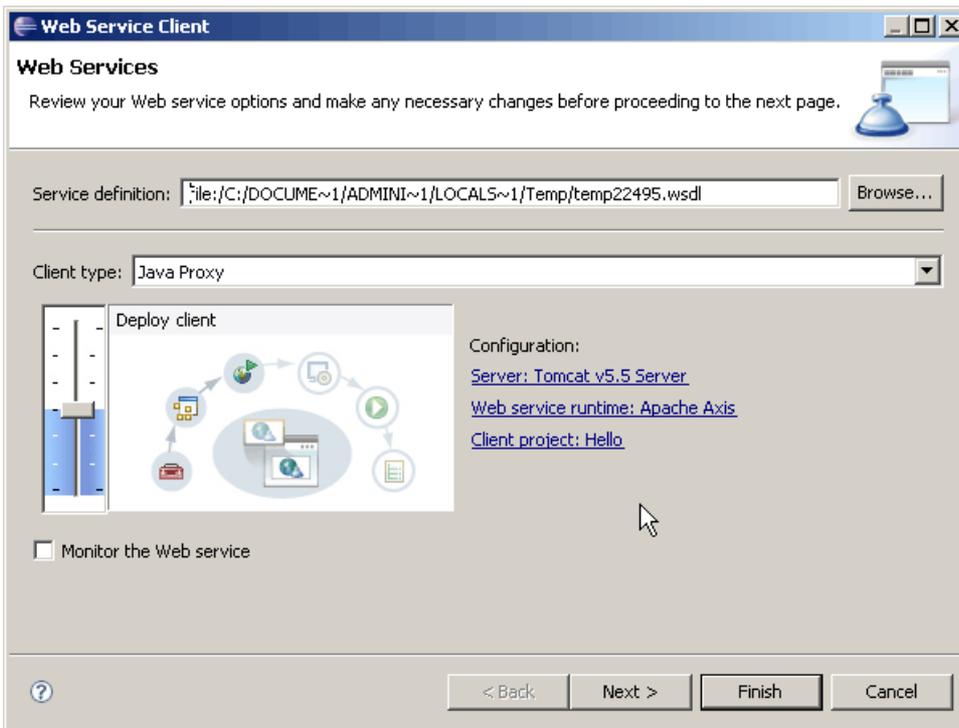
On the 'Details' page of either a Service or a Service Interface, click on the 'Launch Web Service Wizard' link under 'Other Actions' at the bottom of the page. This will display the Wizard's own page (Figure 193).



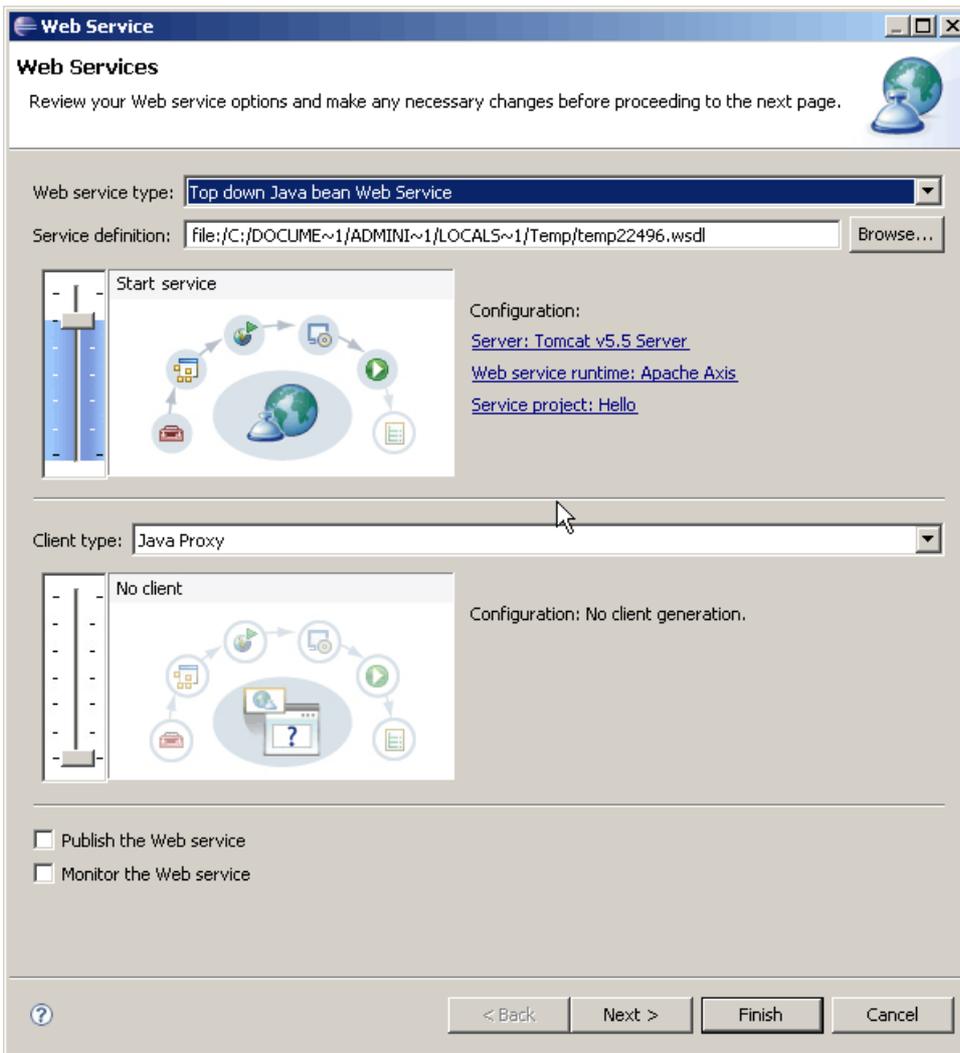
**Figure 193** 'Launch Web Service Wizard' page

The 'Launch' page offers two options, both of which will turn out to be rather familiar in their realisation:

- 'Web Service Client' launches the same client-building wizard that you used earlier in the Part 2 activity *Generating a client from WSDL* (Figure 204).
- 'Web Service Skeleton' launches the same web service creation and client-building wizard you also saw earlier in the Part 1 activity *Implementing a simple web service* (Figure 215).



**Figure 204** Web Service Wizard generating client



**Figure 215** Web Service Wizard generating top-down skeleton web service

The WSDL file is downloaded to a temporary location by Eclipse, which is shown by the 'Service definition' field in both Figure 204 and Figure 215.

The WSDL document is sufficient to generate a client, as you might expect. So following the 'Web Service Client' dialogue, as you have done before, will generate a client that will call the remotely deployed service at the endpoint given in the WSDL.

In the second case (Figure 215), Eclipse is using the WSDL as a basis for generating the actual service 'top down' from the WSDL document (refer back to Figure 1 of *Implementing a simple web service* for the difference between top down and bottom up). However, the WSDL is not, of course, sufficient for Eclipse to implement the actual 'Hello' service. So, if you continue with the 'Web Service Skeleton' dialogue and also, as before, request that a test client is created, the client will be generated but as the service is not actually implemented it will not work.

Eclipse can be used in this way, as the option indicates, to generate a 'skeleton' for the service. If you do this then you should specify a new project name for the service in the dialogue. Then you will see that a set of Java files is created with class names, but these files are simply empty classes that are intended for filling in with the service logic.

Unless you have experience of working with Java you are not recommended to attempt top-down service generation. If you do want to use top-down development, there is a tutorial that may interest you at:

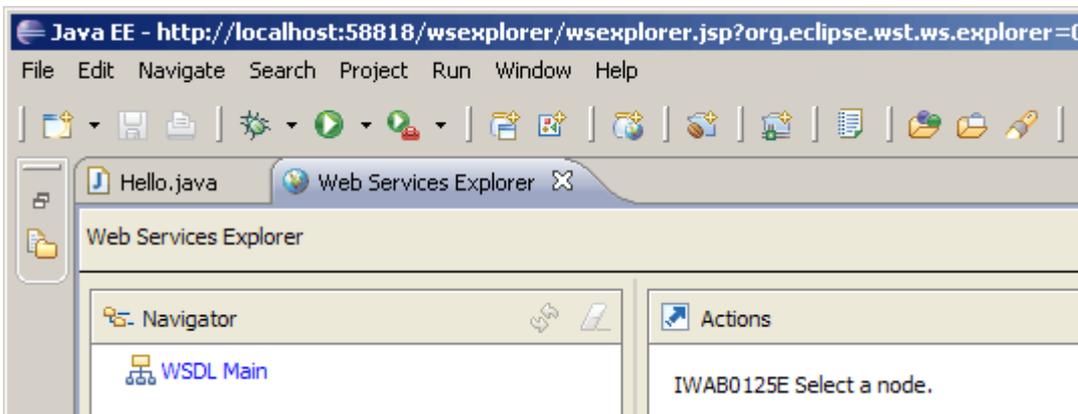
<http://www.eclipse.org/webtools/jst/components/ws/M4/tutorials/TopDownWebService.html>

I don't want to indicate to you that top-down development is more difficult or less favourable than bottom-up. In fact, top-down development is often preferred as it can be used to elevate design to considering 'what' needs to be done, which is described in WSDL, rather than considering the 'how', which is the main concern at the Java coding level. Bottom-up development, which I have used in this block, allowed you to get a quick start in developing and deploying a web service based on simple code that I provided for you, but this may well not be the best choice for other projects.

There is one other way in which the Web Services Explorer allows us to use WSDL, which I shall look at quickly now.

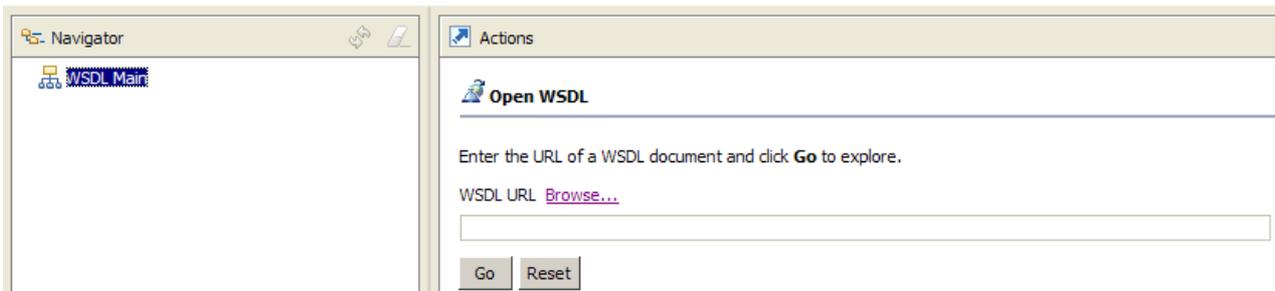
## WSDL Main

As well as providing access to jUDDI, the Web Services Explorer has a 'WSDL mode', itself called the 'WSDL Explorer'. To switch to the WSDL Explorer, click on the 'WSDL Page' icon in the upper right-hand corner (second from the right). This will present the 'WSDL Main' item in the Navigator (Figure 226).



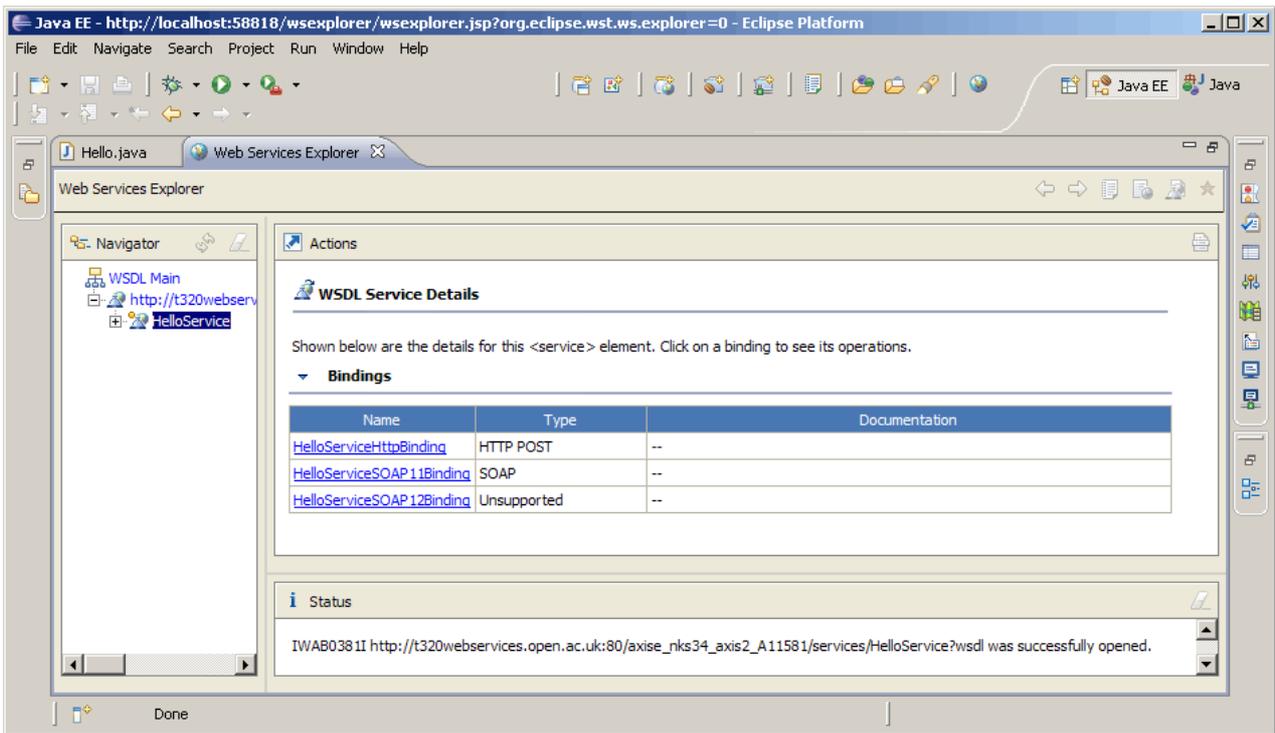
**Figure 226** 'WSDL Main' in Web Services Explorer

If you click on the 'WSDL Main' item, an 'Open WSDL' page will appear (Figure 237).



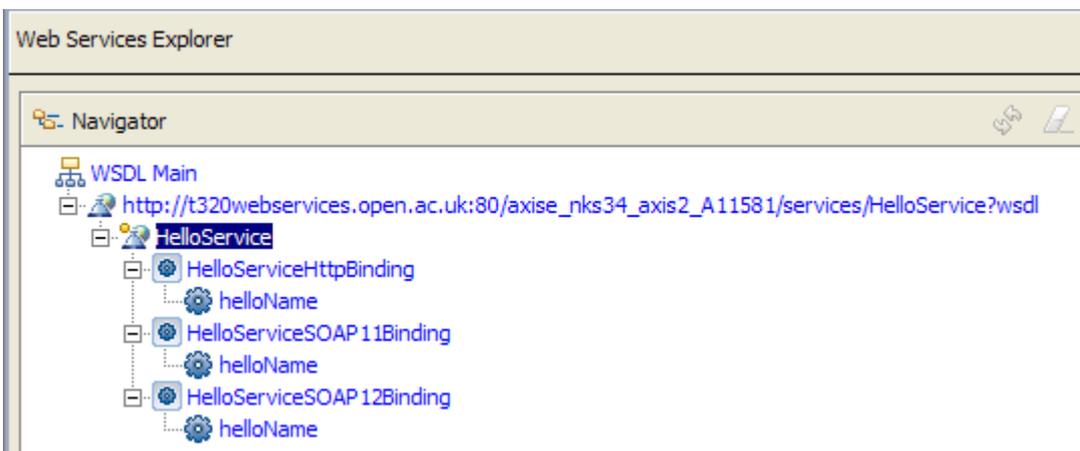
**Figure 237** 'Open WSDL' page

Now type the URL for your own 'Hello' service's WSDL into the 'WSDL URL' field and click 'Go'. You will then see the 'WSDL Service Details' page (Figure 248).



**Figure 248** 'WSDL Service Details' page

The 'WSDL Service Details' page lists bindings from the WSDL. In the Navigator the HelloService is listed, which can be expanded out so that you can view the various components (Figure 29).

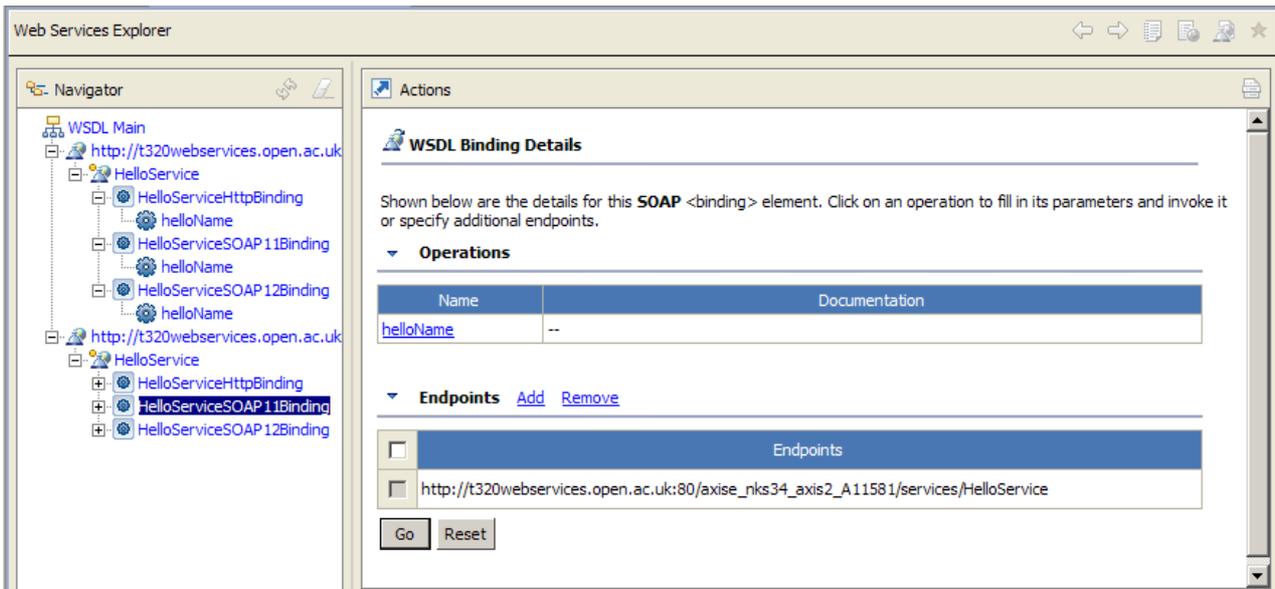


**Figure 29** 'HelloService' fully expanded in Navigator

The Navigator list elements under the HelloService WSDL, which has three different bindings. These are taken directly from the WSDL generated by Axis2 on the server.

The simple service you have deployed, and the Axis2 support on the server side, does not support all of the bindings described in the WSDL. In the WSDL Service Details (Figure 248) you can see that only the HTTP and SOAP11 bindings are supported. The client you are going to use employs SOAP to access a service, so you must use the SOAP version 1.1 'HelloServiceSOAP11Binding' to access the service.

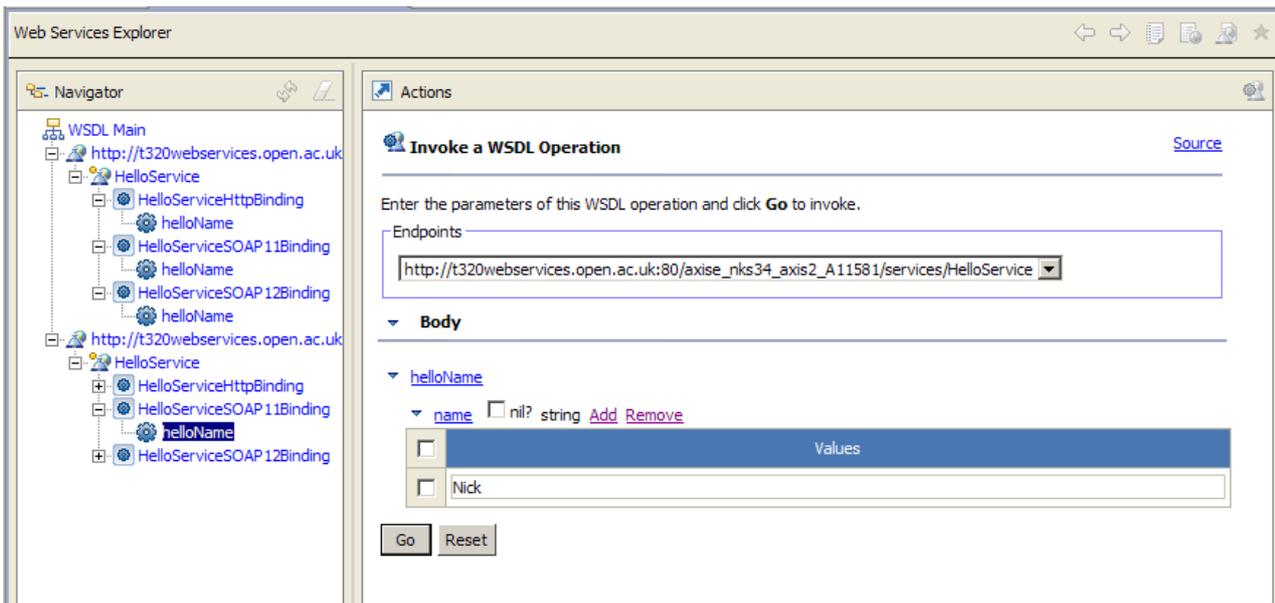
Click on the 'HelloServiceSOAP11Binding' in the WSDL Service Details page so that you see its details, as shown in Figure 250.



**Figure 250** WSDL SOAP 1.1 binding details

Notice that the endpoint is (or should) be that of your own service that you deployed to Axis2.

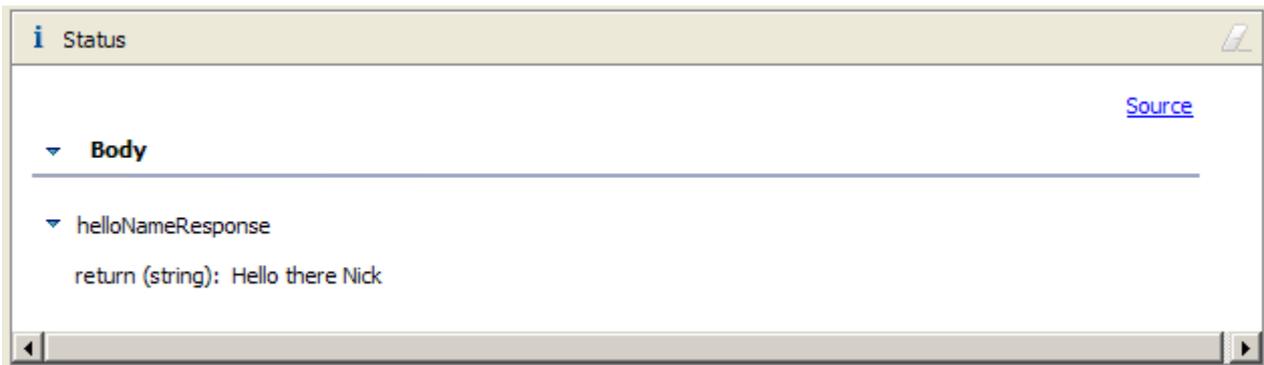
To be able to invoke the service you need to click on the link to the 'helloName' method listed under 'Operations'. This will take you to the 'Invoke a WSDL Operation' page (Figure 261).



**Figure 261** 'Invoke a WSDL Operation' page

Before invoking the service you need to specify the input name value. To do this, tick the checkbox by the 'Values' heading and then click the 'Add' link, which is just above the 'Values' heading (see Figure 261). This will open a textbox field into which you can type a string to be used as the value of 'name' (e.g. 'Nick').

Once you have typed in a name, click on the 'Go' button to invoke the web service. You should see that the result returned by the web service is displayed in the Status window (Figure 272).



**Figure 272** Status window displaying result from the 'Hello' web service

## Summary

In this activity you have seen how to use both the Eclipse WTP and jUDDI to publish and manage UDDI entries. I have also demonstrated how to access a web service based on UDDI entries.

You have been provided with a range of alternative tools and approaches to publishing and accessing services in UDDI or via WSDL. It is up to you to decide which approach you consider 'best', if any, or which you prefer.