

Lesson 5: SOAP, UDDI and WSDL

Lesson 5 serves two purposes. One is to give you an introduction to SOAP and associated topics concerning standards for exchange of information. The other is to show you how easy it is to import documents following these standards, and how you can use the resulting components in your own application. Studying the resulting components can also be a very valuable learning tool.

Component X Studio has a mechanism for browsing the Internet for services available, and for importing their definitions. Once imported (whether from the Internet or from anywhere else), these definitions are used to automatically create all the components described in them. All the work is done for you, and you can use any or all of the components – all you have to do is wire them together. The completed project for this lesson is stored as project fedex in the cxDemos component archive.

Lesson topics include:



- 5.1 Definitions
- 5.2 More on SOAP
- 5.3 More on UDDI
- 5.4 More on WSDL
- 5.5 Starting the FedEx Project
- 5.6 Using the UDDI browser
- 5.7 Importing a WSDL File
- 5.8 Adding the First Set of Components
- 5.9 Adding Another Set of Components
- 5.10 Testing the Finished Application
- 5.11 Review
- 5.12 Challenge Yourself



A Little About SOAP, UDDI and WSDL

It has been said that the full potential of the Internet will not be realized without open standards, and SOAP, UDDI and WSDL are all contributors to open standards. It is not the purpose of this tutorial to fully explain these concepts, but to give you an overview and show you how to use them in building a useful application with Component X Studio. If you are already familiar with them and want to get right into the tutorial, you can skip the next few sections. If you need more details than we give in this tutorial, see the following web sites:

- ❑ SOAP: <http://www.w3.org/TR/SOAP/>
- ❑ UDDI: <http://www.uddi.org/>
- ❑ WSDL: <http://msdn.microsoft.com/xml/general/wsdl.asp>

5.1 Definitions

In case you are not familiar with the acronyms SOAP, UDDI and WSDL, here are their definitions and some brief explanations:

SOAP (Simple Object Access Protocol) is a simple protocol for exchange of information. It is based on XML and consists of three parts: a SOAP envelope (describing what's in the message and how to process it); a set of encoding rules, and a convention for representing RPCs (Remote Procedure Calls) and responses.

UDDI (Universal Description, Discovery, and Integration) is a specification designed to allow businesses of all sizes to benefit in the new digital economy. There is a UDDI registry, which is open to everybody. Membership is free and members can enter details about themselves and the services they provide. Searches can be performed by company name, specific service, or types of service. This allows companies providing or needing web services to discover each other, define how they interact over the Internet and share such information in a truly global and standardized fashion.

WSDL (Web Services Description Language) defines the XML grammar for describing services as collections of communication endpoints capable of exchanging messages. Companies can publish WSDLs for services they provide and others can access those services using the information in the WSDL. Links to WSDLs are usually offered in a company's profile in the UDDI registry.

5.2 More on SOAP

A SOAP (Simple Object Access Protocol) message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The element names are exactly as quoted (XML is case-sensitive), and their rules are as follows:

- The “Envelope” is the top element of the XML document representing the message. It **MUST** be present in a SOAP message, and it **MAY** contain namespace declarations as well as additional attributes. If present, such additional attributes **MUST** be namespace-qualified. Similarly, the element **MAY** contain additional sub elements. If present, these elements **MUST** be namespace-qualified and **MUST** follow the Body element.
- The “Header” element is a mechanism for adding features to a SOAP message without prior agreement between the communicating parties. SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory. It is optional and, if present, it **MUST** be the first immediate child element of an Envelope element. It **MAY** contain a set of header entries, each being an immediate child element of the Header element, and each of which **MUST** be namespace-qualified.
- The “Body” element is a container for information intended for the ultimate recipient of the message. It **MUST** be present in a SOAP message and **MUST** be an immediate child element of an Envelope element. It **MUST** directly follow the Header element if present. Otherwise it **MUST** be the first immediate child element of the Envelope element. The element **MAY** contain a set of body entries, each being an immediate child element of the Body element. Immediate child elements of the Body element **MAY** be namespace-qualified.

Here are some additional rules: An application generating SOAP message **SHOULD** include the proper SOAP namespace on all elements and attributes defined by SOAP. A SOAP application **MUST** be able to process SOAP namespaces in messages that it receives. It **MUST** discard messages that have incorrect namespaces and it **MAY** process SOAP messages without SOAP

namespaces as though they had the correct SOAP namespaces. A SOAP message **MUST NOT** contain a Document Type Declaration and **MUST NOT** contain Processing Instructions.

These rules, and others that we haven't mentioned, are rather abstract and difficult to remember, and you will find that studying the examples is a valuable aid in learning them.

See <http://www.w3.org/TR/SOAP/> for more information.

5.3 More on UDDI

UDDI (Universal Description, Discovery, and Integration) is a specification designed to let businesses find each other and share business information. The UDDI Business Registry is a global, public, online directory that gives businesses a uniform way to describe their services, discover other companies' services, and understand the methods necessary to conduct e-business with a particular company. Initial members include companies as diverse in their activities as Sun Microsystems, Ariba, Boeing, British Telecommunications, CBSI, Merrill Lynch, Descartes, Intel, IBM, Microsoft, and Fujitsu, to name only a very few.

At present there are three UDDI Business Registry operators, IBM, Microsoft and Hewlett-Packard. These companies simply agree to provide the means for companies to register their services, and for others to access them. The data is common to each Registry operator, to provide redundancy and to give worldwide access to everyone as quickly and easily as possible.

Published web services mean developers do not have to “re-invent the wheel” every time they want to do something that others have already done; for example getting a stock quote, tracking an en-route package, reading the news, or checking an airline reservation. Using the Internet, developers can discover and incorporate prewritten web services within their own applications quickly and easily.

Anyone can search the registry, but to publish information about your company and services, you must register. Registration is free, and you do not have to give information you do not want to give. You can opt out of receiving any communications following your registration. Once registered, which takes only a few minutes, you can enter as much or as little about your company and services as you want. You can provide links to web sites, ftp areas, email and postal addresses and more importantly describe the services you provide and methods by which they can be accessed.

See <http://www.uddi.org/> for more information on UDDI.

5.4 More on WSDL

WSDL (Web Services Description Language) is not a new definition language. It simply defines some XML grammar for describing communications regarding web services in a structured and standardized way. It is also extensible and allows using other type definition languages. A WSDL document defines services as collections of network endpoints, or ports, no matter what message formats or network protocols are used to communicate. Ports are capable of exchanging

messages, which are defined as abstract descriptions of the data being exchanged. A WSDL message uses the following elements to define web services:

- **Types**– containers for data type definitions using some type system (such as XSD).
- **Message**– an abstract, typed definition of the data being communicated.
- **Operation**– an abstract description of an action supported by the service.
- **Port Type**–an abstract set of operations supported by one or more endpoints.
- **Binding**– a concrete protocol and data format specification for a particular port type.
- **Port**– a single endpoint defined as a combination of a binding and a network address.
- **Service**– a collection of related endpoints.

The **types** element encloses data type definitions that are relevant for the exchanged messages. It does not require that the XSD system be used (although that is preferred.) It defines the types in a message whether or not the resulting wire format is actually XML. An extensibility element may appear under the **types** element to identify the type definition system being used and to provide an XML container element for the type definitions.

Messages consist of one or more logical **parts**, which are flexible mechanisms for describing the logical abstract content of a message. Each part is associated with a type from some type system using a message-typing attribute.

Operations are one of the following four types of transmission:

- **One-way**. The endpoint receives a message.
- **Request-response**. The endpoint receives a message, and sends a correlated message.
- **Solicit-response**. The endpoint sends a message, and receives a correlated message.
- **Notification**. The endpoint sends a message.

A **Port Type** is a named set of abstract operations and the abstract messages involved. The port type **name** attribute provides a unique name among all port types defined within in the enclosing WSDL document.

WSDL also defines a common **binding** mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. This allows the reuse of abstract definitions. WSDL currently provides specific binding extensions to SOAP, HTTP GET /POST and MIME formats and protocols, but any other binding mechanisms can also be used.

A **port** defines an individual endpoint by specifying a single address for a binding.

A **service** groups a set of related ports together.

Although all those definitions look complicated, once you see an actual example, it doesn't look so bad. Figure 5-1 shows the WSDL we will be using in the FedEx tracking application.

```

<?xml version="1.0" ?>
- <definitions name="FedExTrackerService"
  targetNamespace="http://www.xmethods.net/sd/FedExTrackerService.wsdl"
  xmlns:tns="http://www.xmethods.net/sd/FedExTrackerService.wsdl"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
- <message name="statusRequest">
  <part name="trackingNumber" type="xsd:string" />
</message>
- <message name="statusResponse">
  <part name="return" type="xsd:string" />
</message>
- <portType name="FedExTrackerPortType">
  - <operation name="getStatus">
    <input message="statusRequest" name="getStatus" />
    <output message="statusResponse" name="getStatusResponse" />
  </operation>
</portType>
- <binding name="FedExTrackerBinding" type="FedExTrackerPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  - <operation name="getStatus">
    <soap:operation soapAction="urn:xmethodsFedEx#getStatus" />
    - <input>
      <soap:body use="encoded" namespace="urn:xmethodsFedEx"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    - <output>
      <soap:body use="encoded" namespace="urn:xmethodsFedEx"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
- <service name="FedExTrackerService">
  <documentation>Provides access to a variety of FedEx delivery status
  information</documentation>
  - <port name="FedExTrackerPort" binding="FedExTrackerBinding">
    <soap:address location="http://services.xmethods.net:80/perl/soaplite.cgi" />
  </port>
</service>
</definitions>

```

Figure 5-1: The WSDL file for the FedEx Tracking Application

See <http://msdn.microsoft.com/xml/general/wsdl.asp> for more information on WSDL.

5.5 Starting the FedEx Project



1. You need to be connected to the Internet for this tutorial, in order to import the WSDL file, and in order to get information from the FedEx web site.
2. If Component X Studio is not already running, start it as before and create a new project. Add a new package, and name it `fedex`.

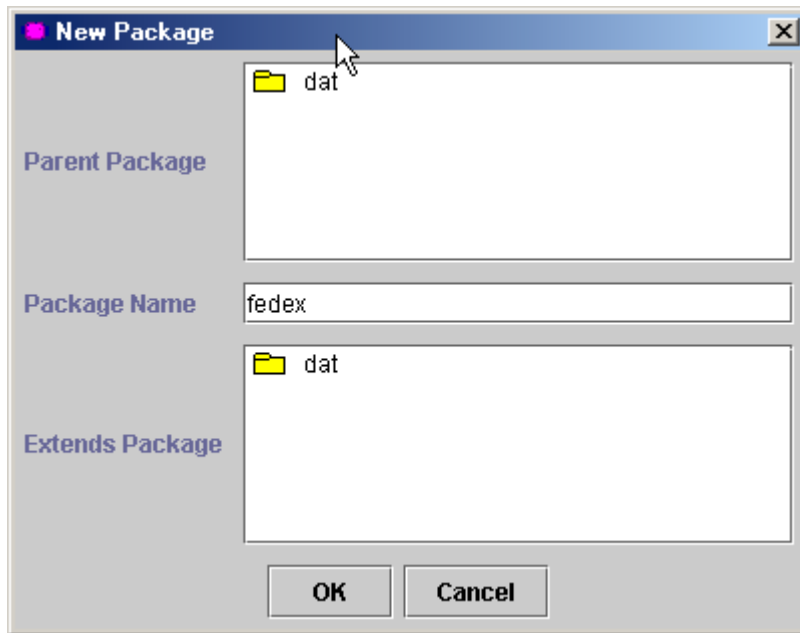


Figure 5-2: The New Package dialog

5.6 Using the UDDI browser



1. From the Project Menu, select **Import | UDDI Browser**.
2. Enter x and click OK
3. From the list of companies presented, select xMethods and click OK.
4. You will see a list of services as shown in Figure 5-3. If we were going to use this method of importing the WSDL file, and if everything works correctly, the WSDL file referenced at the UDDI registry for the business and service you selected would be imported into your project and all the necessary components would be created. However, things don't always go according to plan, and it may be that the referenced file is corrupted or not available for some reason. In that event, and if you know the name of the WSDL file, you can access it directly, as we will see in the next section. Don't worry about importing the same file twice – only one set of components will be built.
5. Click **Cancel** as often as necessary to back out of the UDDI import.

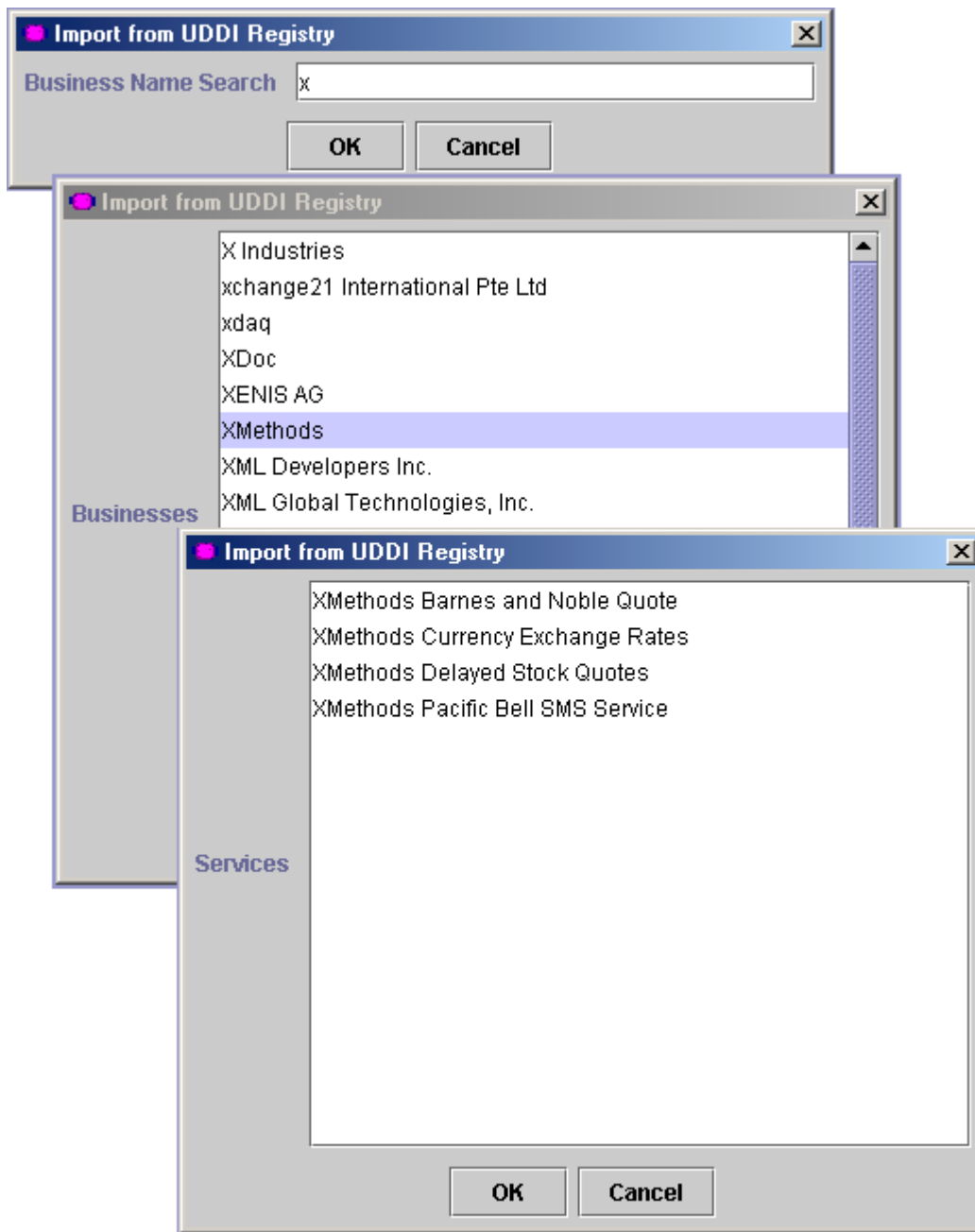


Figure 5-3: Import from UDDI Registry

5.7 Importing a WSDL File



1. From the Project Menu, select **Import | WSDL file**.
2. As shown in Figure 5-4, make sure the **fedex** package is selected and enter the following URL: `http://www.abacus-labs.com/fedextrackerservice.WSDL`. This particular URL is simply a parking space for a copy of a distilled version of the FedEx WSDL file so that we can be sure you will be able to find it. Click OK.

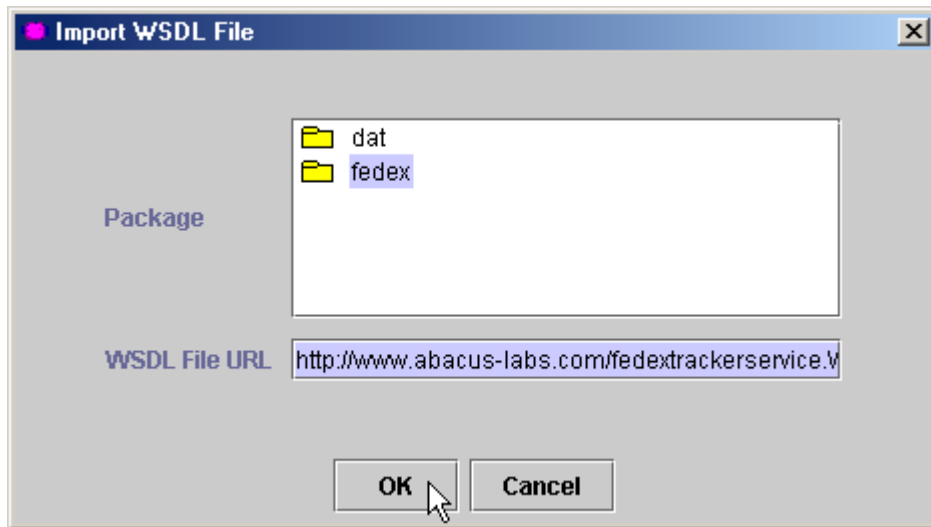


Figure 5-4: Entering a WSDL file URL

3. Once the WSDL file has been imported, you will find that a number of component definitions have been built and added to your project, ready for you to use in your application. This is the power of importing a WSDL file—once you have identified a service that you want to use, you simply import it and all the components defined are built for you. Figure 5-5 shows the components built for you and added to your project as soon as you imported the WSDL file, either directly or by way of a UDDI browse.

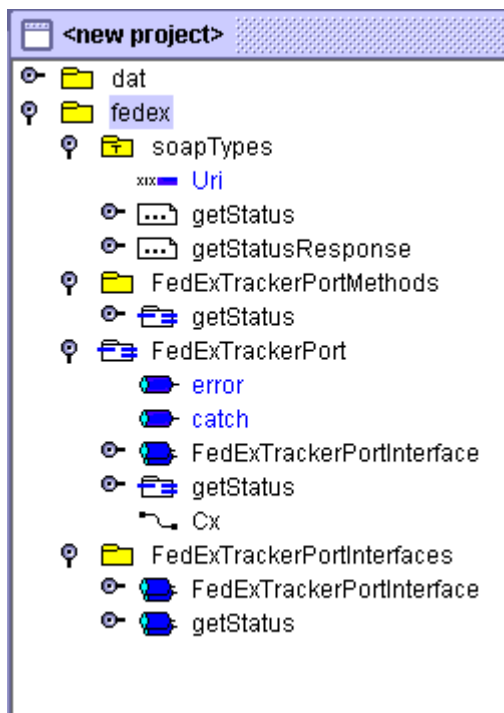


Figure 5-5: The component tree after importing a WSDL file

5.8 Adding the First Set of Components



1. Add a new component to the package and name it `track1`
2. As shown in Figure 5-6, add the following components:

A `getStatus` interface to the right side of the new component

A `getStatus` document to the MIDDLE of the new component (so it does not become a port.)

A dialog from the utility palette

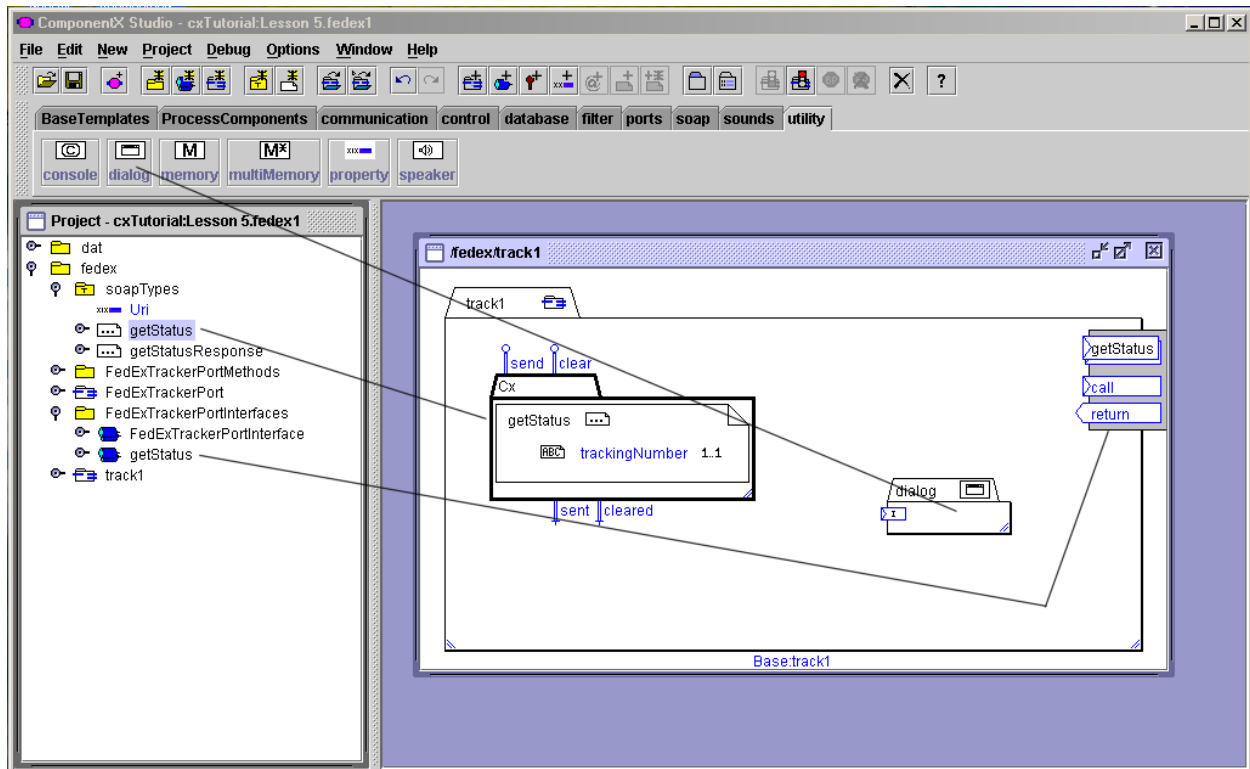


Figure 5-6: Adding components

3. As shown in Figure 5-7, wire these components as indicated:
 - The output of the `getStatus` document to the call port of the interface
 - The return port of the interface to the input of the dialog

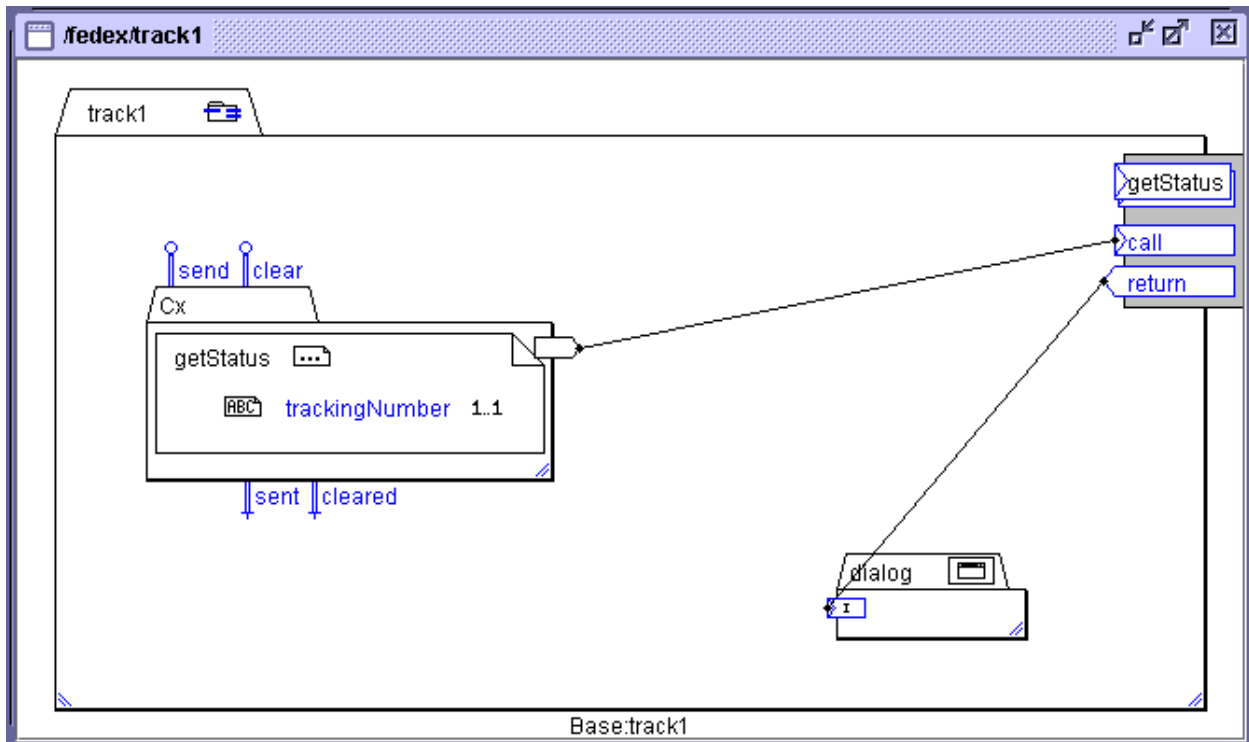


Figure 5-7: Wiring components together

4. Make sure the document is selected (showing a dark line around it), and right-click to open the initial value dialog.

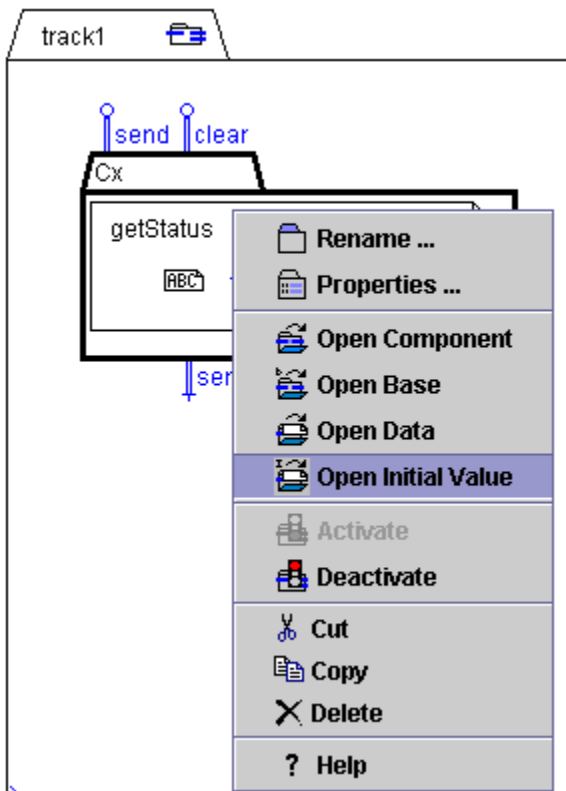


Figure 5-8: Opening the Initial Value Dialog

- Click in the Value column for `trackingnumber` and enter the number 791901881310. Click Enter, then click OK
- Add a pin named “send” to the track1 component and connect it to the send pin of the `getStatus` document

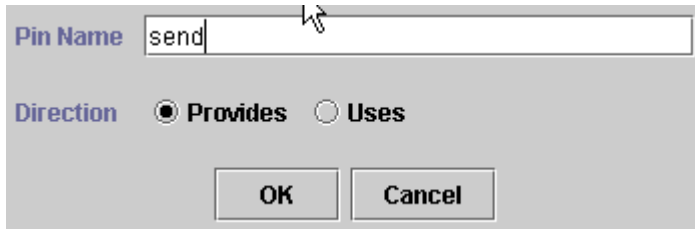


Figure 5-9: Adding a send pin

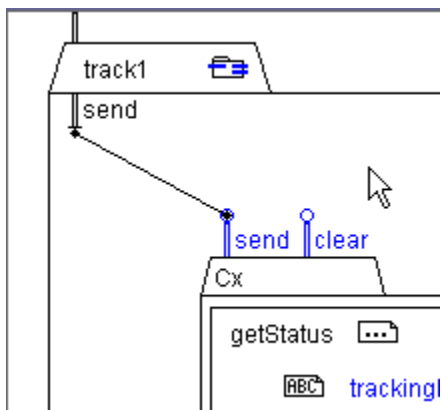


Figure 5-10: Connecting the send pin

5.9 Adding Another Set of Components



- Close track1 and make a new component named track2
- Select track1 and drag it into the new track2
- Add a send pin to track2 and wire it to the send pin of track1
- Select `fedextrackerportinterface` and drag it to the right side of track2
- Wire the `getStatus` interfaces together. You should end up with a component looking like Figure 5-11

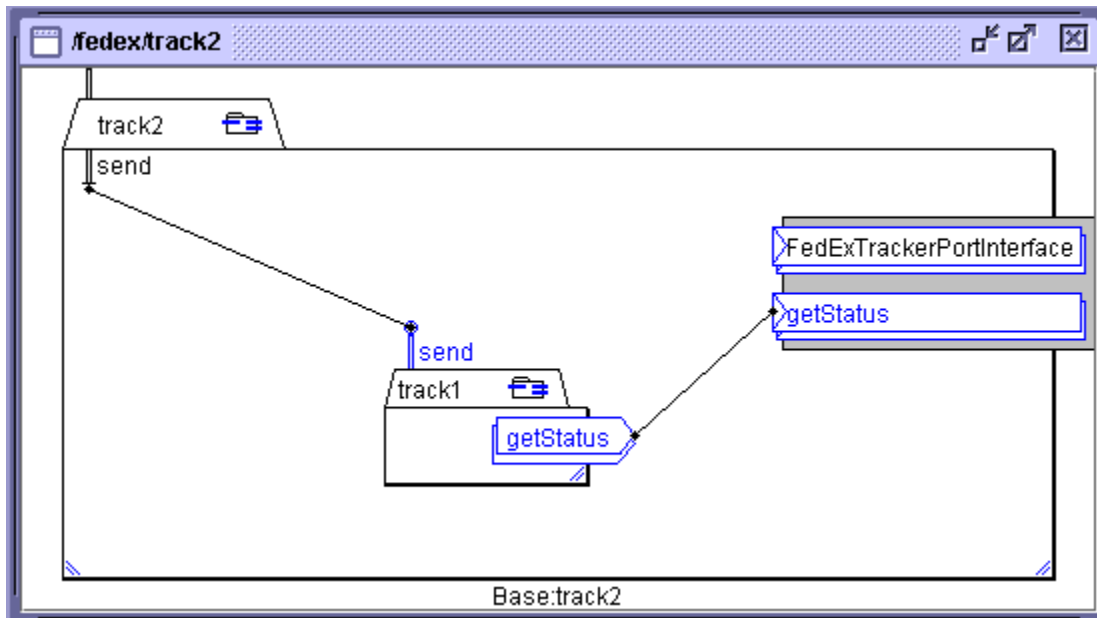


Figure 5-11: Adding component Track2

6. Close track2 and make a new component named track3
7. Select track2 and drag it into the new track3
8. Select fedextrackerport and drag it into track3
9. Wire the last two components you added together. You should end up with a component looking like Figure 5-12

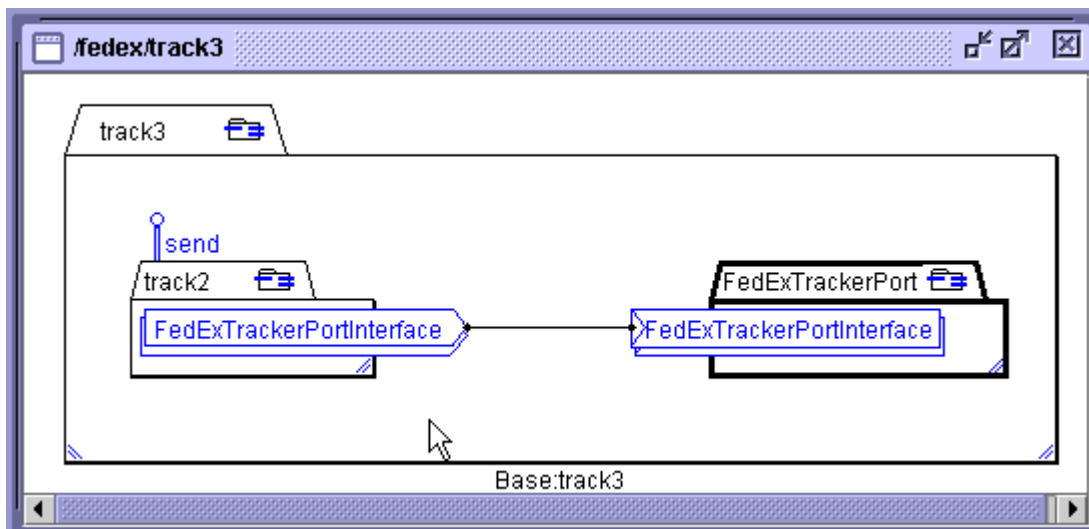


Figure 5-12: Adding component Track3

5.10 Testing the Finished Application



1. Pulse (double-click) the pin of the track2 component and a dialog with a send button appears.
2. Click the send button, and within a few seconds, the FedEx tracking status should appear as shown in Figure 5-13. Notice that the information is in an XML message, and you can use the information in your own applications in any way you want.

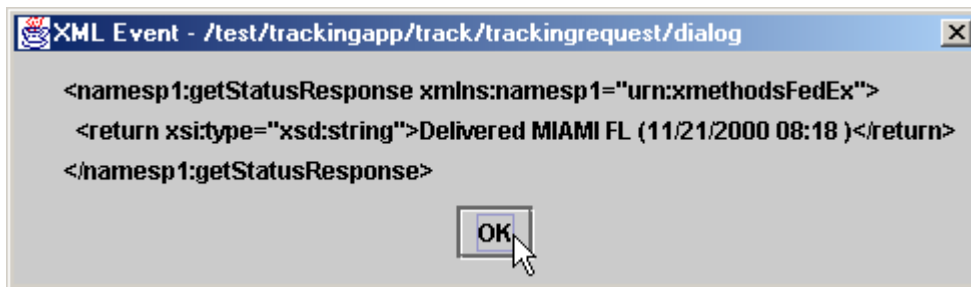


Figure 5-13: A Successful Response

If something goes wrong, you will see an error returned as in Figure 5-14 and you can learn quite a lot from this XML message. Not only will you see the cause of the failure, but you can also see information sent so you can check it for accuracy.

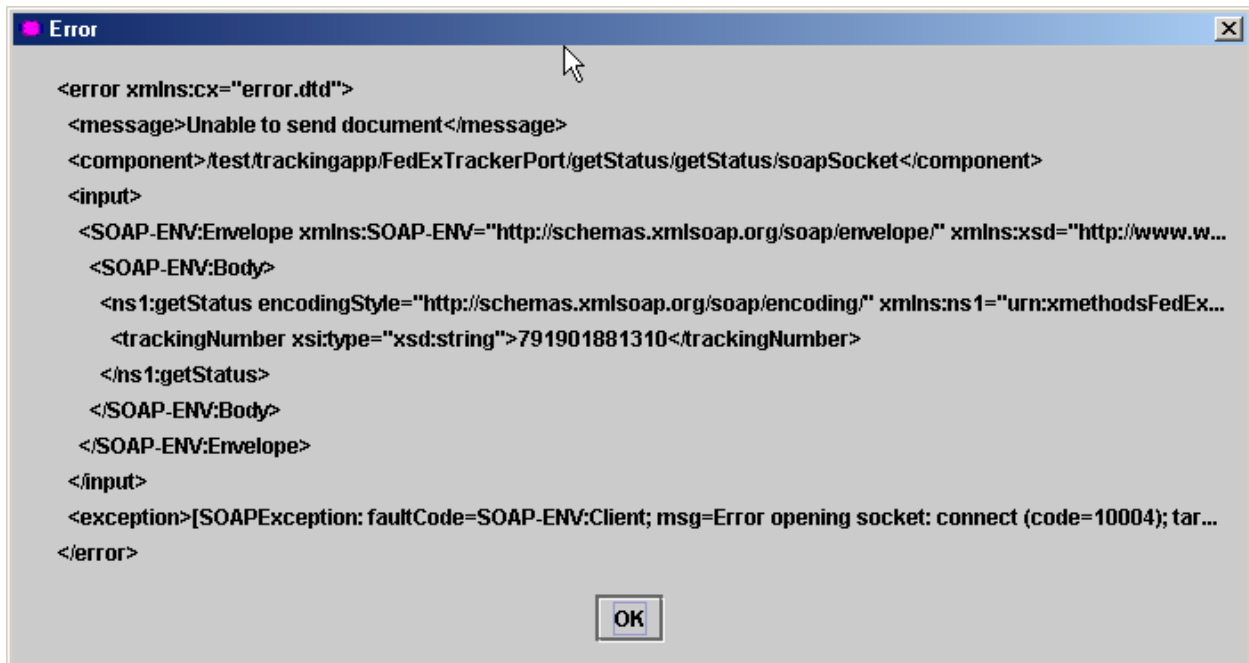


Figure 5-14: An Unsuccessful Response

5.11 Review



In this lesson you learned:

1. What SOAP is
2. What UDDI is
3. What WSDL files are
4. How to browse for businesses and services you are interested in, and importing their WSDL files.
5. How to import a WSDL file directly from the Internet and have Component X Studio make all the component definitions for you
6. How to add instances of the component documents and interfaces to a project.
7. How to add components inside each other and wire them together.
8. How to learn from both successful and unsuccessful XML messages returned.

5.12 Challenge Yourself



Test what you learned in this lesson by browsing for companies and services you might be interested in, and import a service to a new project. For example, you could import *Delayed Stock Quotes* from the services offered by *xMethods*, then build an application just like the FedEx tracking one. Instead of an initial value of a tracking number, enter a stock ticker symbol, and the result will be a 20-minute delayed quotation for that stock.