Build Web services with PHP in Eclipse

Using PHP development tools for contract-first development

Skill Level: Intermediate

Nathan A. Good (mail@nathanagood.com)
Senior Information Engineer
Consultant

01 Jul 2008

Learn how to build Web services in PHP using the PHP Development Tools (PDT) plug-in in Eclipse in three easy steps. First, become familiar with the PDT project, and learn how to create and deploy useful PHP projects. Second, learn the philosophy behind contract-first development. Finally, get an informative overview of the basic parts that make up a Web Services Description Language (WSDL) file.

Section 1. Before you start

About this tutorial

This tutorial shows how to build Web services in PHP using the PHP Development Tools (PDT) plug-in. The PDT project was released in its 1.0 release version in September 2007 and was followed by a V1.0.2 release in January 2008. The PDT project gives you first-class abilities to edit, debug, and deploy PHP applications in the Eclipse IDE.

Objectives

This tutorial has three main objectives. The first is to become familiar with the PDT project and learn how to create and deploy useful PHP projects. The second is to

learn about the philosophy behind contract-first development. Third, this tutorial serves as an informative overview of the basic parts that make up a Web Services Description Language (WSDL) file.

Prerequisites

You should have experience with PHP development.

System requirements

To get the most out of this tutorial, you need to install Eclipse and the PDT plug-in. You must also install the Eclipse Web Standard Tools (WST) subproject.

Note: If you installed the Java™ 2 Platform, Enterprise Edition (J2EE) bundle of Eclipse, you already have WST.

To follow the examples, you need Eclipse and one of the operating systems that Eclipse supports — Mac OS X, Microsoft® Windows®, or Linux®. You also need a Java Runtime Environment (JRE) — at least JRE for Java 5 is recommended.

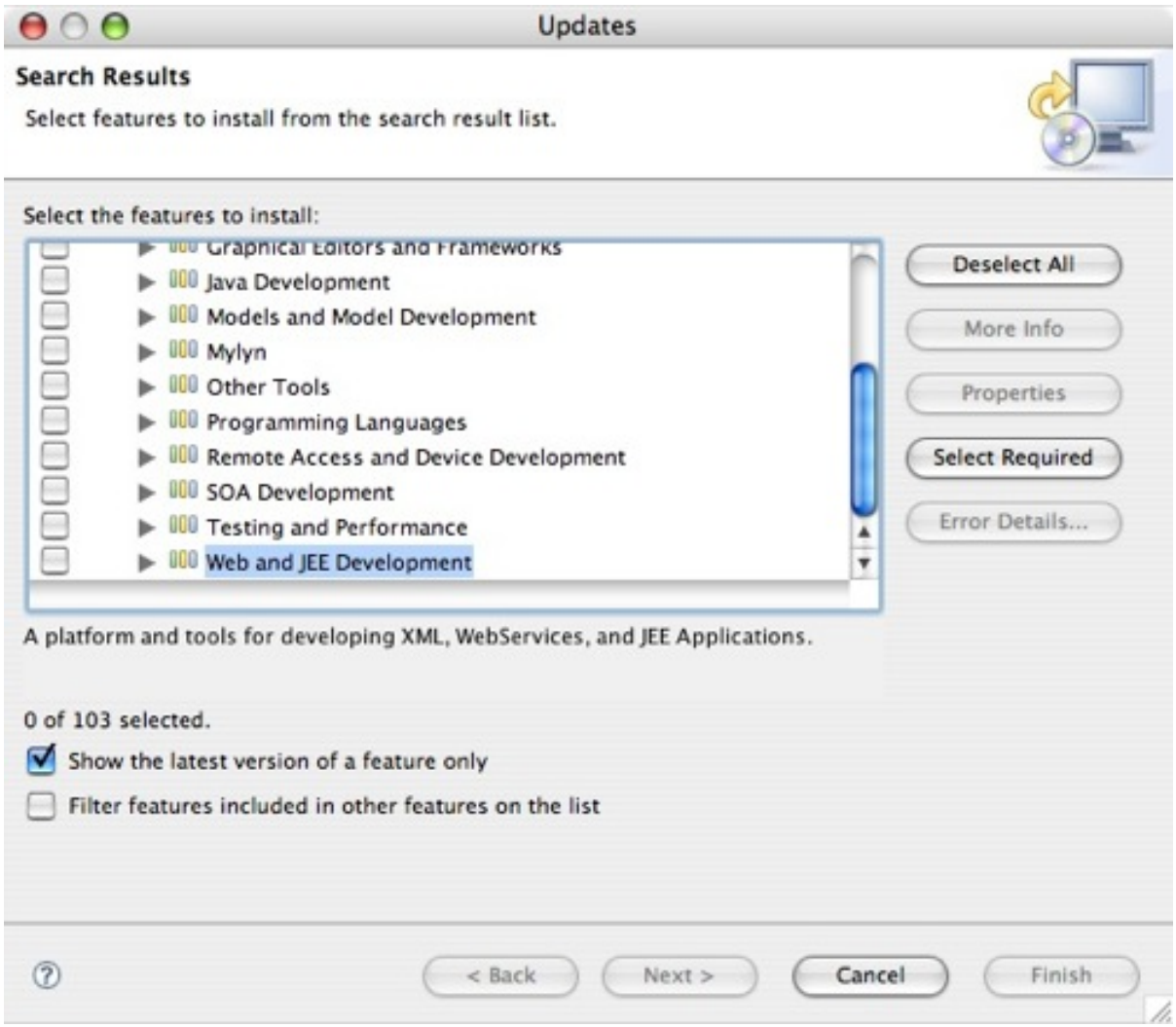
Section 2. PDT overview

The PDT project gives you the ability to do PHP development using the Eclipse IDE. It includes many features of the Java editing environment, including syntax highlighting, code templating, perspectives, and file and project wizards.

Install PDT

To install PDT, make sure you have the latest version of Eclipse. Use the built-in software updater to update PDT from the update site. PDT requires the WST subproject, so install WST first if you don't already have it. If you downloaded and installed the J2EE bundle of Eclipse, you already have WST. If you downloaded the Java-only bundle of Eclipse, you must install WST from the Eclipse discovery site.

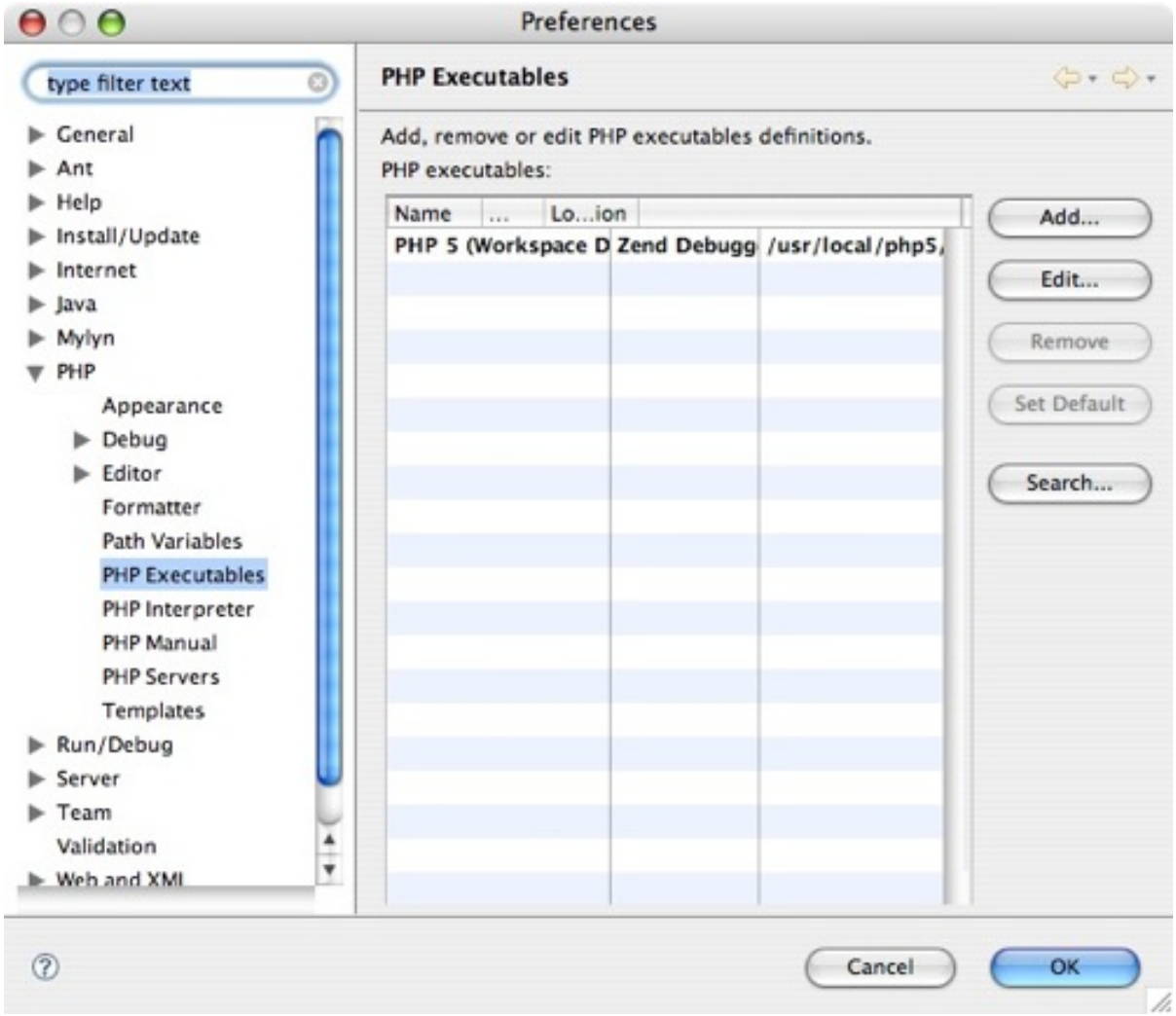
Figure 1. Installing WST from the Eclipse discovery site



Set up PDT

After installing PDT, there are a couple things you might want to set up, especially if you're using PDT and Eclipse on a computer on which you've already been doing some PHP development. The first configuration is the path to the PHP executable. To set this path, open the Preferences window, and in the left pane, expand **PHP**, then click **PHP Executables**. In the right pane, you'll see where you can type or browse to the path of your PHP executable file.

Figure 2. Setting up the PHP interpreter



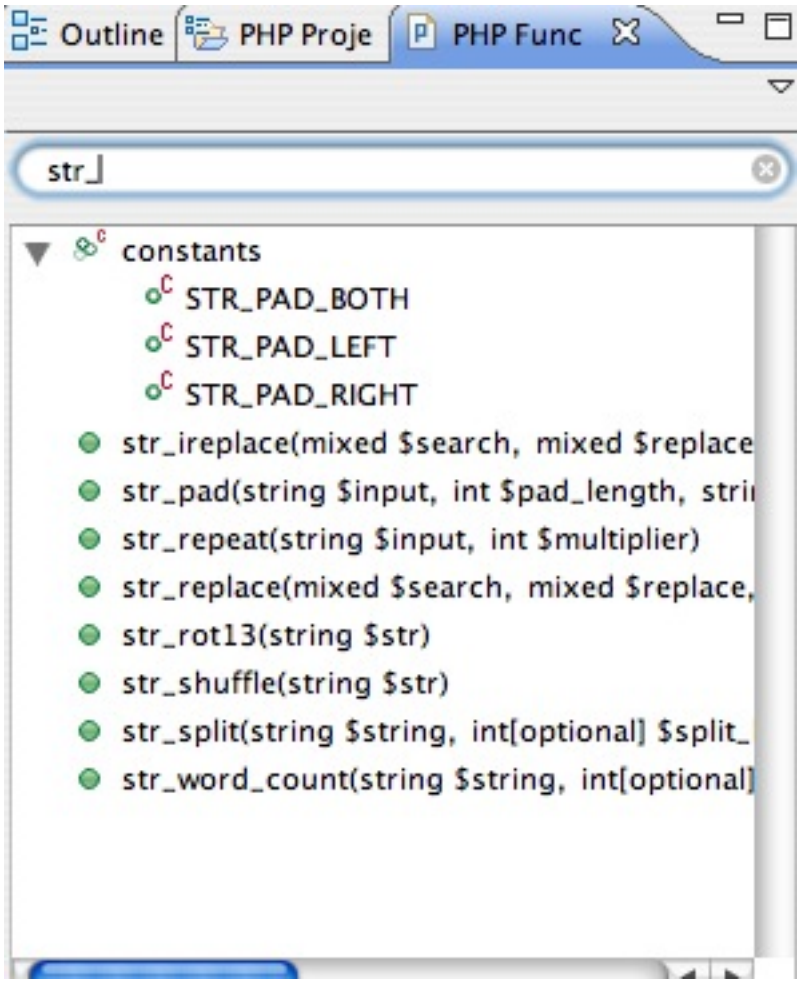
Under **PHP** in the left pane, click **PHP Interpreter**, then, specify the version of PHP you're using. The default is PHP V5.

To access the PHP editor, under **PHP** in the left pane, expand **Editor**. The PHP editor that comes with PDT includes many features of the Java editor, including syntax highlighting, formatting, code completion, and code templates.

The PHP Functions tab

The **PHP Functions** tab shows the common PHP functions and classes available in your library path. You can search for functions in the list by starting to type in the search box, as shown in Figure 3. Double-click the name on the **PHP Functions** tab to insert the name into the editor at the current cursor position. You can also use code completion (**Ctrl+Space**) when typing in the editor to pick from a list of matching functions or templates and insert them.

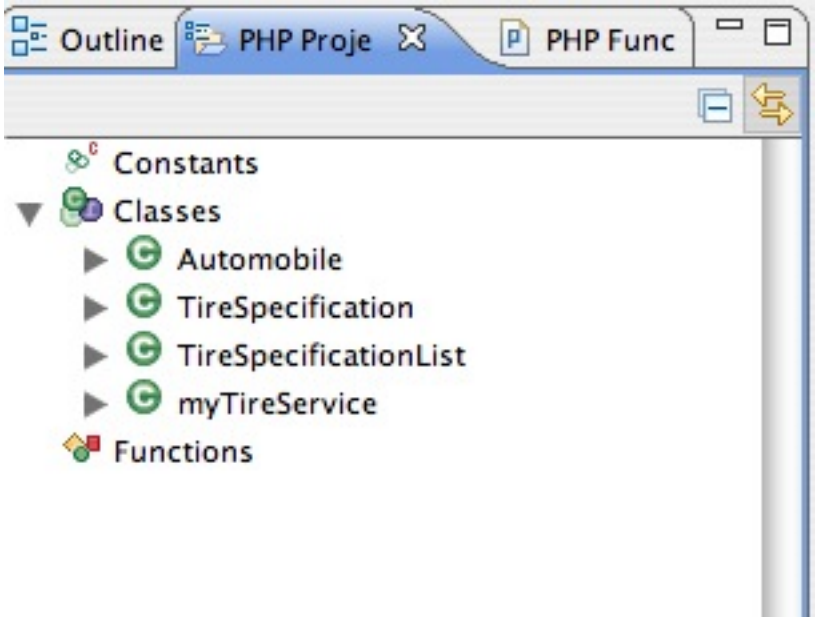
Figure 3. Searching for PHP functions



The PHP Project tab

The **PHP Project** tab provides a PHP-centric view of your PHP project by listing the contents of your project split into object types. You see any classes you have defined in your project under **Classes** and any functions under **Functions**. Double-click the item on the **PHP Project** tab to automatically navigate to the item in the editor in which it's found. Figure 4 shows an example of the **PHP Project** tab with the classes and functions discussed later.

Figure 4. The PHP Project tab



The PHP Explorer

The PHP Explorer shows a list of the projects and files in your workspace. It's consistent with the other project explorers, such as the Java Explorer.

Section 3. Building a service contract

WSDL overview

A WSDL file is an XML file that describes one or more Web services. The description includes everything from the methods, or *operations,* available in the service to the schema for the messages that form the inputs and outputs of the operations.

If you're familiar with WSDL files or already have one that you want to use and don't care how they work, you can safely skip this section and move on to "Alternative: Import a WSDL file." But if you're new to WSDL files or have used them before, but would like a refresher, this section will be very helpful.

The ability to visually build WSDL files and XML Schema Definition (XSD) files in Eclipse should be a great help when building WSDL files in Eclipse. If you'd like, you can always switch the WSDL editor to Source mode to edit the source yourself. The WSDL file includes four major parts, which are described at a high level below and

explored in greater detail in the rest of this section.

`types`
This section of the WSDL file covers the types used in the Web service request and response messages.

`message`
This contains the messages sent to and received from the Web service. In the visual Eclipse editor, they are shown as the input and output of the Web service operation.

`portType`
This lists the operations the Web service exposes.

`binding`
This includes the communication protocols the Web service uses.

`service`
This describes the service or services exposed.

The types element

The `types` element can contain information about the types used in the SOAP messages. The types are conveniently described in XML schema, and you can import other types. The Eclipse editor works well with both. One advantage of using external schemas is that you can reuse complex types that have already been defined. In contrast, an advantage of using inline (that is, inside the WSDL) schema is that your WSDL file contains everything that needs to be known about the Web service.

Note: To keep things simple, this tutorial focuses on defining types in inline schema.

The message element

The `message` element defines the messages sent in and received from the operations. If you think of a Web service operation as being a method in PHP code, the `message` element describes the parameters sent into the method and the return type that the method returns.

The portType element

The `portType` element describes the operations a Web service exposes. If you think of a Web service, again, as a PHP class, the `portType` element would be explaining the public methods the PHP class exposes.

The binding element

The `binding` element explains how the operations are exposed in terms of which protocol is used to access them. In most of the examples you find, the SOAP protocol is used to access Web service operations. However, other protocols may be used, such as HTTP.

The service element

The `service` element describes the Web service, allowing you to group similar operations in a collection of operations that make sense as a service.

Contract-first development

Building or providing a WSDL file first, then writing the code to implement the service according to the WSDL file (contract) is called *contract-first* development. Building the contract before worrying about how you're going to implement it has a few advantages.

Open implementation
: It leaves you free to implement the service however you see fit. As long as you comply with the contract, it doesn't matter if you build the Web service in the PHP, Java, Groovy, or any other language.

Concurrent development
: If you build the contract first, you can give the contract out to your consumers so they can start developing clients to call the services. As long as you comply with the contract you've given to your consumers, you both can build your code concurrently. If you did bottom-up Web service development, in which you build the code first, then expose the contract for the service, other people are waiting for you to finalize your code base.

Proper analysis
: Creating the contract first keeps you and your team honest by performing analysis up front because the analysis leads to the content of the contract. If you build the Web service from the bottom up, there's a greater likelihood that your service grows organically, and you could end up with a contract more specific to your implementation than you might intend.

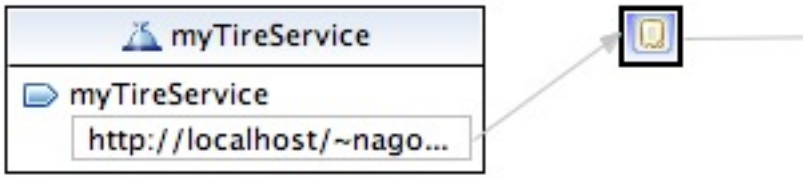
Section 4. Building the WSDL file visually

Build the file in the WSDL visual editor

When you build a WSDL file visually in the Eclipse WSDL visual editor, it feels natural, given the visual editor layout, to build the WSDL file starting at the `service` element and working backward, defining the types last. The editor first shows you a `UI` element that actually corresponds to the `service` element in the WSDL file.

Figure 5 shows part of the Web service. Because the example is a Web service that returns the manufacturer's recommended tire sizes given the year, model, and make of an automobile, call the service *myTireService.* You can type the new name directly into the visual editor.

Figure 5. The Web service in the WSDL file



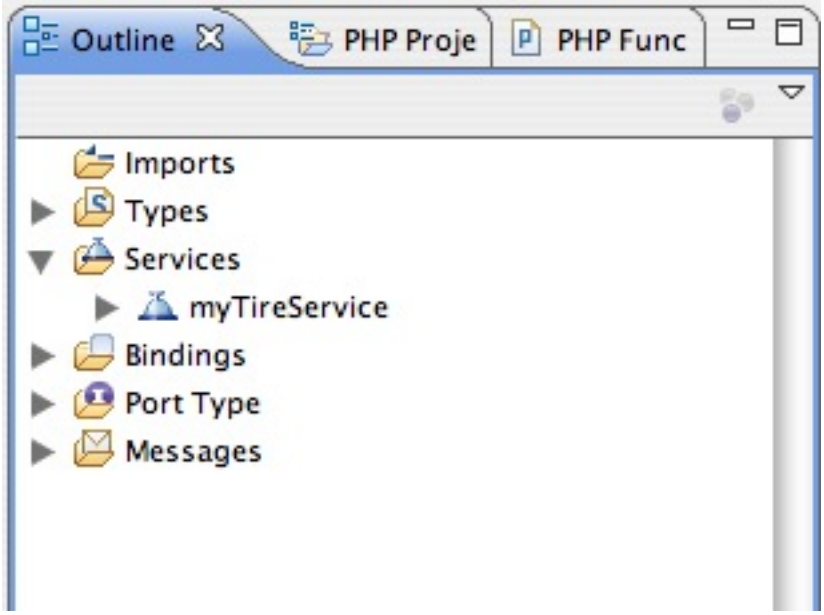
Listing 1 shows the change Eclipse makes for you in the WSDL source.

Listing 1. The WSDL changes to service

```
<wsdl:service name="myTireService">
        <wsdl:port binding="tns:myTireServiceSOAP"
                name="myTireService">
                <soap:address
                        location="http://localhost/~nagood/phpws/service.php" />
        </wsdl:port>
</wsdl:service>
```

The **Outline** tab displays the major parts of the WSDL file. As you click the parts on the **Outline** tab, they're highlighted in the WSDL visual editor.

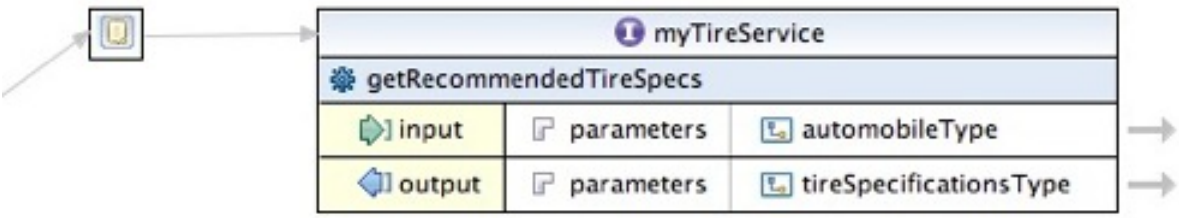
Figure 6. The Outline tab



Update the name of the default operation

After changing the name of the service, you can update the name of the default operation that was created when Eclipse initially built the skeleton WSDL file. Click **NewOperation** and start typing the name of the method. The name of the method in this example is `getRecommendedTireSpecs`, as shown in Figure 7. The updated WSDL source is shown in Listing 2.

Figure 7. The operation in the WSDL editor

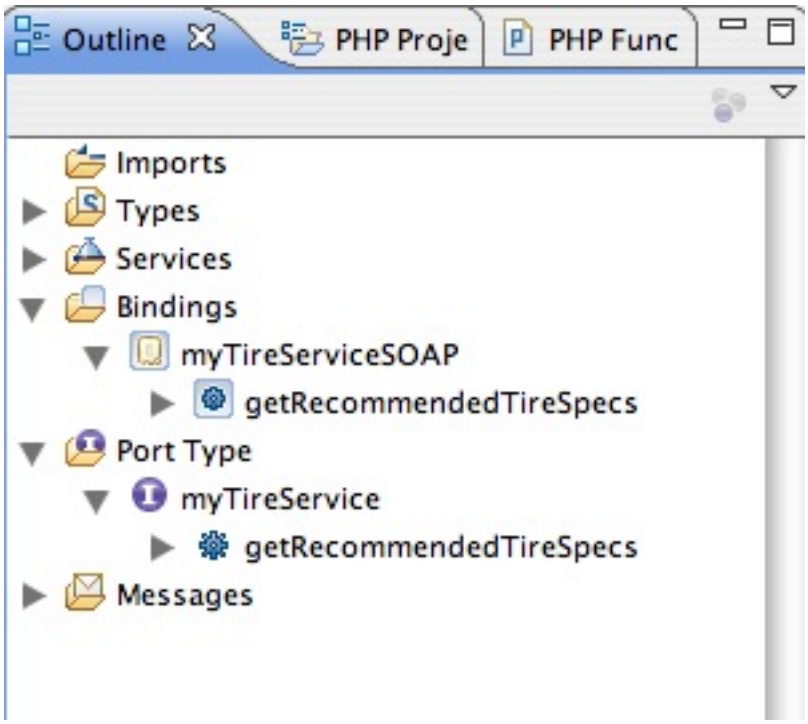


Listing 2. Updated WSDL source for getRecommendedTireSpecs

```
<wsdl:portType name="myTireService">
        <wsdl:operation name="getRecommendedTireSpecs">
                <wsdl:input message="tns:getRecommendedTireSpecsRequest" />
                <wsdl:output message="tns:getRecommendedTireSpecsResponse" />
        </wsdl:operation>
</wsdl:portType>
```

The **Outline** tab is updated along with the visual editor, as shown in Figure 8.

Figure 8. The Outline tab after editing the operation

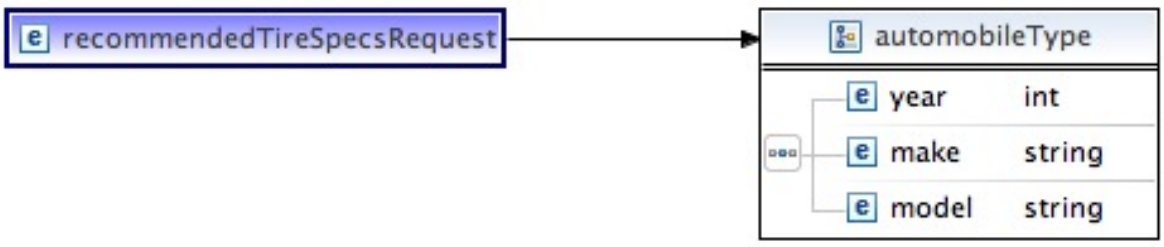


Leave the binding alone for now. It's defined as it should be, and there's no reason to change it. However, in this example, the types of the request and response will be augmented from the default type of strings.

Edit request and response types

On the input line of the WSDL visual editor, click the blue arrow next to the generated request element name `getRecommendedTireSpecs`. The Eclipse visual XML schema editor opens for you automatically, allowing you to edit the complex types for the request and response. The request is shown in Figure 9, with the elements added for the year, make, and model of the automobile.

Figure 9. The getRecommendedTireSpecs request object



Right-click the type, then click **Refactor > Make Anonymous Type Global** to be able to reuse the complex type elsewhere in the WSDL file. A global type is not necessary for purposes of this example, but it's helpful to see the difference. Listing

3 shows the type after it was made into a global type.

Listing 3. After making the type a global type

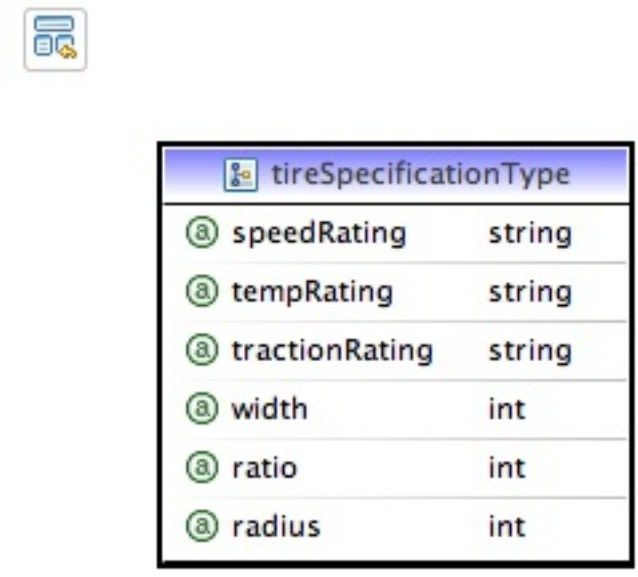
```
<!-- ... parts of the WSDL before ... -->
<xsd:element name="recommendedTireSpecsRequest" type="tns:automobileType"></xsd:element>
<xsd:complexType name="automobileType">
                <xsd:sequence>
                        <xsd:element name="year" type="xsd:int"></xsd:element>
                        <xsd:element name="make" type="xsd:string"></xsd:element>
                        <xsd:element name="model" type="xsd:string"></xsd:element>
                </xsd:sequence>
</xsd:complexType>
```

Go back to the WSDL editor and click the blue arrow next to the output line. The schema editor opens once again, this time with the element and anonymous type displayed. Because the example service returns one or more sets of tire specifications, each consisting of the speed rating, traction rating, temperature rating, and dimensions of the tire, the simple string in the return type is not adequate.

Add the data parts as attributes

This time, instead of adding elements to the complex type, add the different data parts as attributes. Remove the element that's there, then add attributes for `speedRating`, `tempRating`, `tractionRating`, `width`, `ratio`, and `radius`.

Figure 10. After adding the attributes



Add another complex type called `tireSpecificationsType`. This is just a

collection of `tireSpecificationType` complex types.

Figure 11. The tireSpecificationsType

Now change the `recommendedTireSpecsResponse` to make it of type `tireSpecifcationsType`. When you're done, it looks like Figure 12. Now, you have all the types you need to use the Web service. The final WSDL file looks like Listing 4.

Figure 12. The final response

Listing 4. The completed WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="urn:myTireService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="myTireService"
    targetNamespace="urn:myTireService">
    <wsdl:types>
        <xsd:schema targetNamespace="urn:myTireService">
            <xsd:element name="recommendedTireSpecsRequest"
                type="tns:automobileType">
            </xsd:element>
            <xsd:element name="recommendedTireSpecsResponse"
                type="tns:tireSpecificationsType">
            </xsd:element>
            <xsd:complexType name="automobileType">
                <xsd:sequence>
                    <xsd:element name="year" type="xsd:int"></xsd:element>
                    <xsd:element name="make" type="xsd:string"></xsd:element>
                    <xsd:element name="model" type="xsd:string"></xsd:element>
                </xsd:sequence>
            </xsd:complexType>

            <xsd:complexType name="tireSpecificationType">
                <xsd:attribute name="speedRating" type="xsd:string"></xsd:attribute>
                <xsd:attribute name="tempRating" type="xsd:string"></xsd:attribute>
                <xsd:attribute name="tractionRating"
                    type="xsd:string">
                </xsd:attribute>
                <xsd:attribute name="width" type="xsd:int"></xsd:attribute>
                <xsd:attribute name="ratio" type="xsd:int"></xsd:attribute>
```

```
                <xsd:attribute name="radius" type="xsd:int"></xsd:attribute>
            </xsd:complexType>

            <xsd:complexType name="tireSpecificationsType">
                <xsd:sequence>
                    <xsd:element name="tireSpecification"
                        type="tns:tireSpecificationType" minOccurs="0"
                        maxOccurs="unbounded">
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="getRecommendedTireSpecsRequest">
        <wsdl:part name="parameters" type="tns:automobileType" />
    </wsdl:message>
    <wsdl:message name="getRecommendedTireSpecsResponse">
        <wsdl:part name="parameters" type='tns:tireSpecificationsType' />
    </wsdl:message>
    <wsdl:portType name="myTireService">
        <wsdl:operation name="getRecommendedTireSpecs">
            <wsdl:input message="tns:getRecommendedTireSpecsRequest" />
            <wsdl:output message="tns:getRecommendedTireSpecsResponse" />
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="myTireServiceSOAP" type="tns:myTireService">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="getRecommendedTireSpecs">
            <soap:operation
                soapAction="urn:myTireService#getRecommendedTireSpecs" />
            <wsdl:input>
                <soap:body use="literal" namespace="urn:myTireService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal" namespace="urn:myTireService"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="myTireService">
        <wsdl:port binding="tns:myTireServiceSOAP"
            name="myTireService">
            <soap:address
                location="http://localhost/~nagood/phpws/service.php" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Section 5. Validating the Web service

Alternative: Import a WSDL file
If you don't need to build a WSDL file for your Web service because you already have one, you can import the WSDL file into Eclipse. After doing so, you can still validate it or see it visually using the WSDL visual editor.

Validate the Web service in Eclipse

After you're finished with the WSDL file, validate it inside Eclipse to make sure it's a valid WSDL file. Choose **Validate** from the context menu of the project in PHP Explorer to validate the file with Eclipse. As long as the file is valid, you're ready to start building the PHP Web service.

Section 6. Building the PHP service code

Using SOAP extensions

The SOAP extension in PHP allows you to build SOAP clients and servers in PHP. To build the SOAP client, you use primarily the `SoapClient` class in PHP. In addition to the `SoapClient` class, there are classes to build SOAP servers (`SoapServer`) and classes for dealing with SOAP messages (`SoapHeader`, `SoapFault`, `SoapParam`, and `SoapVar`). For more information about these classes, see the links to the PHP SOAP documentation in Resources.

In V4 of PHP, the SOAP extensions were an optional addition. To use SOAP extensions, you had to either enable the extensions with the `--enable-soap` option or look for precompiled distributions of PHP that already had the SOAP extension enabled. In PHP V6, the SOAP extension is enabled by default.

SOAP extensions with PHP
If you're unsure whether you have SOAP extensions enabled for your version of PHP, there are a couple ways to tell. The first is to write a quick PHP file containing only `<?php phpinfo(); ?>` and view it in a browser or execute it from the command line. Alternatively, you can use the command-line option `-m`, which shows the compiled-in modules for PHP.

As long as you have SOAP extensions enabled, you can create a `SoapServer` dynamically, without having to worry about serializing your PHP objects to XML or deserializing them from XML to your PHP classes.

The Web service used in this tutorial demonstrates how to build a PHP Web service with a slightly more complicated WSDL and class structure over a standard `echo` or Hello World service that takes a string and returns a string. To build this service, the

PHP code must include some objects that are mapped to the complex types in the WSDL file by the SOAP extension code.

The supporting classes

In the PHP service code, although you don't have to worry about serializing and deserializing objects to and from XML, you should build classes that correspond to the types in the WSDL file. There are projects written in PHP that help with this a bit, but with only three types in this service, the effort is fairly trivial.

Build the Automobile class

The first class to build in the PHP code is the `Automobile` class. It actually maps to the `automobileType`, but thanks to the ability to specify the mapping while creating the PHP `SoapServer`, you don't have to make the names of the classes in PHP map to the names of the complex types in the WSDL file. This can make the PHP code look more like PHP code, as it's common to follow Pascal or upper-camel naming conventions for classes in PHP. In contrast, if you've done some surfing on the Web, you'll see many examples in which people keep the names of the PHP classes the same as the complex types in the WSDL file, presumably because it makes it easier to remember what's associated with what.

The `Automobile` PHP class is shown in Listing 5. The fields exposed as elements in the WSDL file are public fields in the class. Note the absence of accessors in the class.

Listing 5. The Automobile PHP class

```
class Automobile {
    public $year; // int
    public $make; // string
    public $model; // string
}
```

Build the TireSpecification class

The second class is the `TireSpecification` class. In the type definition in the WSDL file, the fields for `speedRating`, `tempRating`, and others are attributes. In the PHP class, they're still defined as public fields, just like in the `Automobile` class in Listing 5.

Listing 6. The TireSpecification class

```
class TireSpecification {
    public $speedRating; // string
    public $tempRating; // string
```

```
    public $tractionRating; // string
    public $width; // int
    public $ratio; // int
    public $radius; // int
}
```

Listing 7. The WSDL definition for automobileType

```
<xsd:complexType name="automobileType">
<xsd:sequence>
        <xsd:element name="year" type="xsd:int"></xsd:element>
        <xsd:element name="make" type="xsd:string"></xsd:element>
        <xsd:element name="model" type="xsd:string"></xsd:element>
</xsd:sequence>
</xsd:complexType>
```

Build the TireSpecificationList class

Finally, the `TireSpecificationList` class contains one field, which holds an array of `TireSpecification` objects. This maps back to the `tireSpecificationsType` in the WSDL file and will be the value returned from the function that handles the Web service.

Listing 8. The TireSpecificationList class

```
class TireSpecificationList {
    public $tireSpecification; // tireSpecificationsType
}
```

The complex types all have corresponding PHP classes, so the code is nearly complete. If you look at the **PHP Project** tab, you will see the list of new classes. All that's left in the code is to add a class to be the service and some code to tell the `SoapServer` how to handle the WSDL.

Build the service class

The service class is a lightweight class that contains the method mapping to the operation name in the WSDL file. The operation is shown in Listing 9. Note that the names are the same and that the function takes a single parameter, which will be an `Automobile` object.

Listing 9. The WSDL operation

```
<wsdl:binding name="myTireServiceSOAP" type="tns:myTireService">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getRecommendedTireSpecs">
    <soap:operation
        soapAction="urn:myTireService#getRecommendedTireSpecs" />
    <wsdl:input>
        <soap:body use="literal" namespace="urn:myTireService"
```

```
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal" namespace="urn:myTireService"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
```

To keep things simple, I hard-coded a couple of tire specifications for my car, a 1993 Nateauto Speedster (OK — that's a really corny name, but I'd rather be accused of having a poor imagination for marketing than trademark infringement). I made up a rule that if the car is older than 1994, a smaller tire is among the recommend tire sizes.

Add the server code

Finally, the server code is listed at the end of your service.php file. The `SoapServer` is instantiated with the class map and a couple of options.

Listing 10. The SoapServer code

```
$server = new SoapServer('myTireService.wsdl',
array('classmap' => $classmap,
    'soap_version' => SOAP_1_2,
    'uri' => 'urn:myTireService',
    'style' => SOAP_RPC,
    'use' => SOAP_LITERAL));

$server->setClass('myTireService');
$server->handle();
```

Table 1 shows the options for creating this code.

Table 1. Options for creating SoapServer

Key name	What it does
classmap	An associative array that links the complex type names from the WSDL file to the PHP class names
soap_version	The version of the SOAP specification that `SoapServer` supports
uri	The Universal Resource Identifier (URI) used as the namespace for the service (You will see `urn:myTireService` in the WSDL file.)
style	The style of WSDL method binding used (In this example, **RPC literal** is used.)
use	The type of binding to use, which can be one of SOAP_LITERAL ("literal") or SOAP_ENCODED ("encoded")

The `myTireService` class is set so that when the `handle()` method handles the post, the `getRecommendedTireSpecs()` function is called.

Section 7. Deploying the code

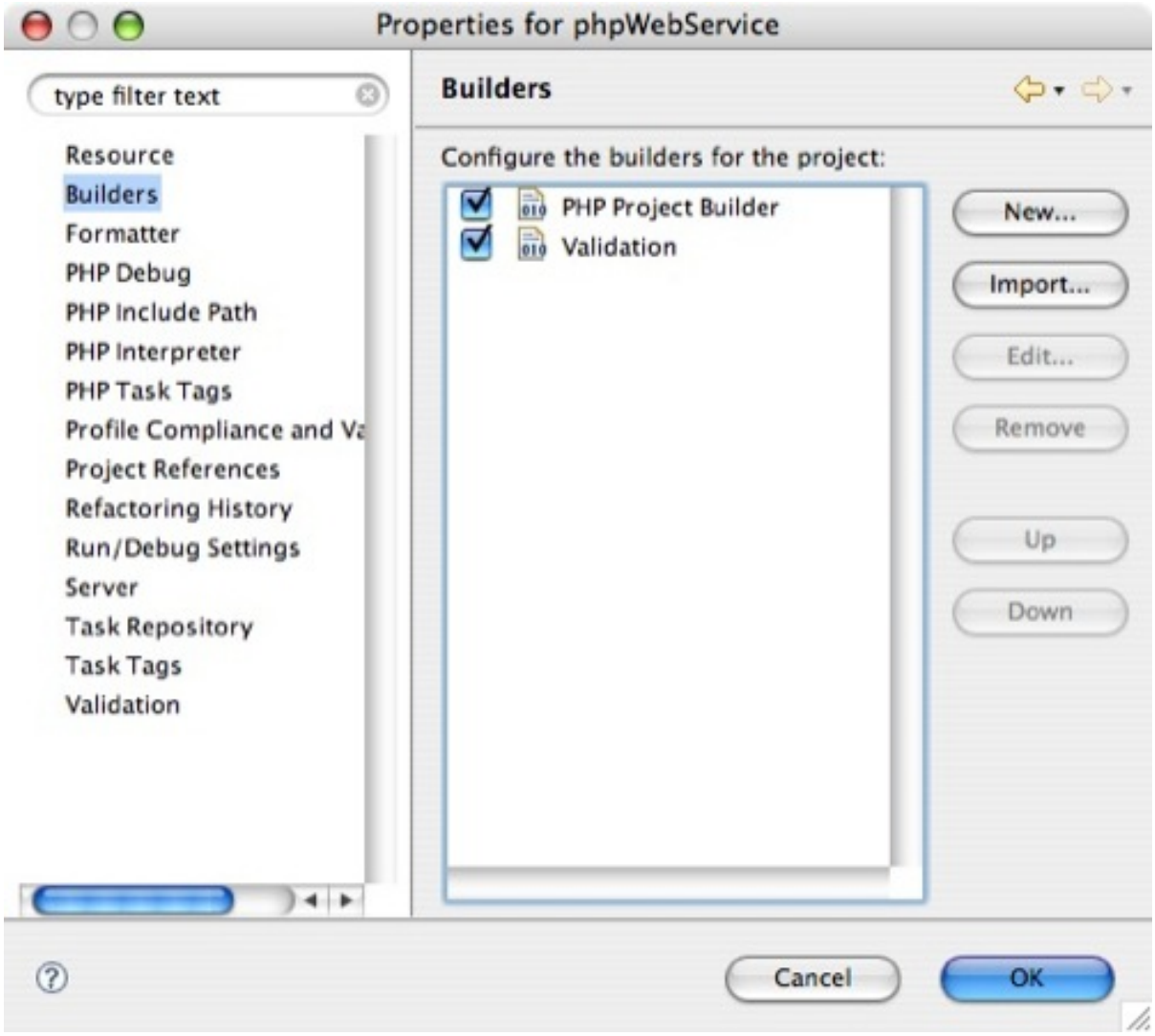
Add a builder for your project

Although the PDT tools allow you run PHP scripts and execute PHP Web pages given a server runtime, neither one helps too much with testing Web services because you need to post a SOAP request to the Web service and be able to see the response. So, the goal is to be able to publish the service.php file and myTireService.wsdl file to a Web location from which you can call them from a test SOAP client.

You really shouldn't put your workspace under the document root or a folder published by the Web server because of permissions and security. The best alternative is to put a builder into Eclipse that publishes your files to the Web document location of your choice. To add a new builder for your project, click **Properties** from the context menu for your project.

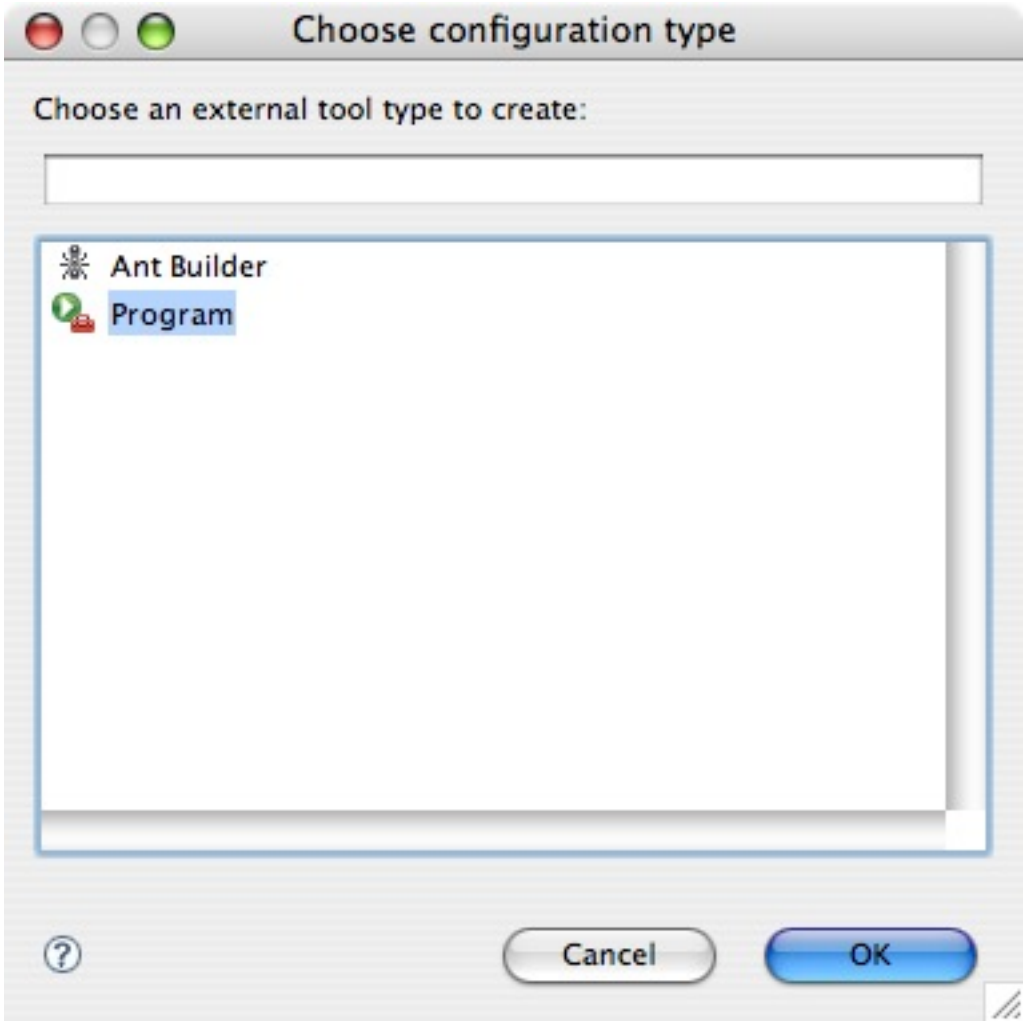
In the left pane, click **Builders**. Click **New** to add a new builder for your project.

Figure 13. Adding a new builder



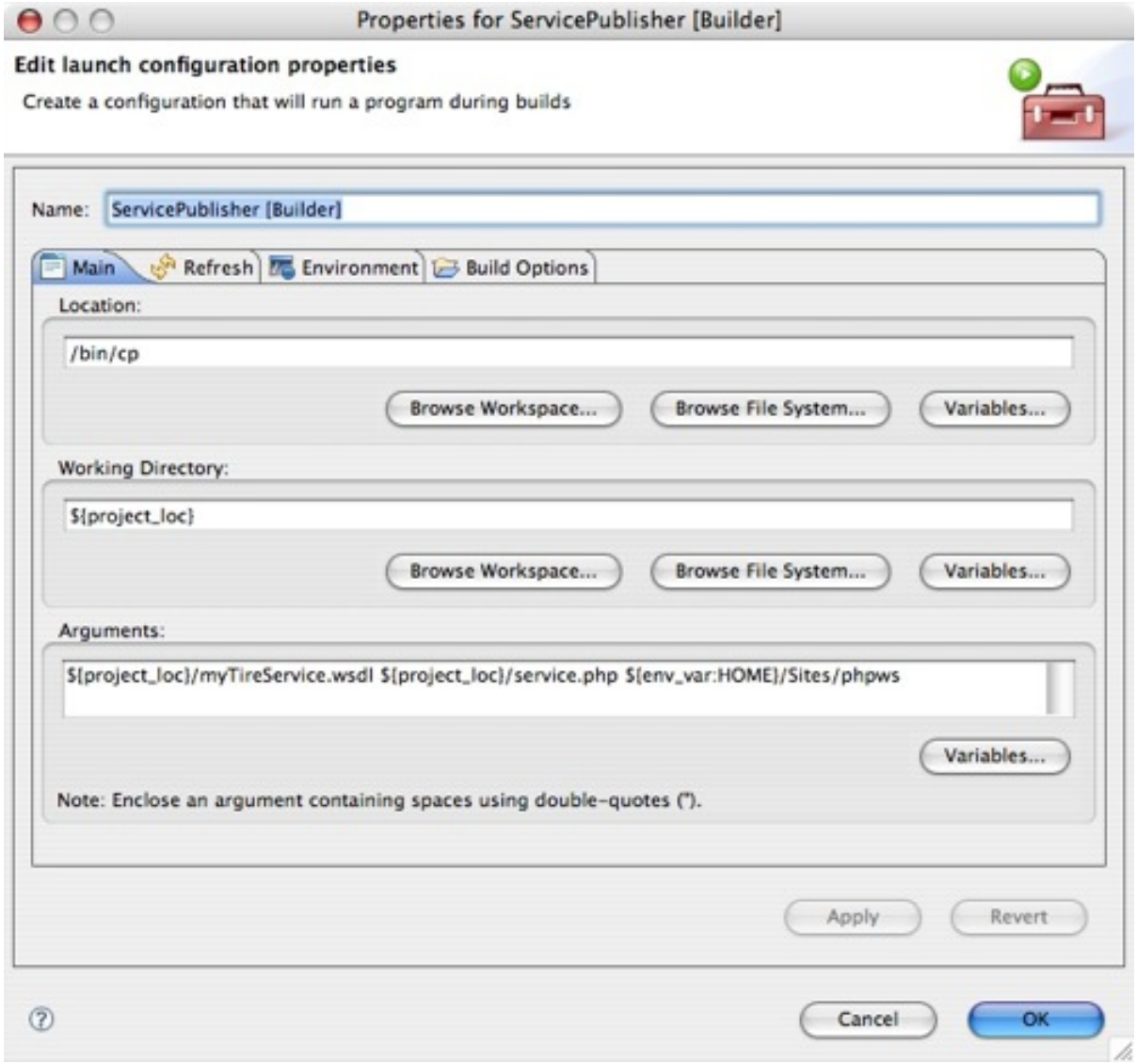
In the **Choose configuration type** window, select **Program** from the list, then click **OK**.

Figure 14. The list of available builders



Type the name of a builder — for example, *ServicePublisher.* In the **Location** field, type the name of a copy command appropriate for your operating system, like `/bin/cp`.

Figure 15. Configuring the new builder to deploy your service



Beneath the **Arguments** field, click **Variables**, then select the variable called `${project_loc}` as the base of your project. Remember that you will have to type the names of the actual files in this **Arguments** field. You can't use wildcards like the asterisk (*) or question mark (?) because these are interpreted by your shell before being sent to the command line. As far as `/bin/cp` is concerned, there is no file called * or ?.

Troubleshooting

Another alternative: REST services
Representational State Transfer (REST) is an architecture or style of using Web services, not a standard like SOAP. REST Web services are simple and can use any other standard, like HTTP or XML. The Internet (basically) is a huge collection of REST services,

where you make the request by supplying a URL to a server and receive a document as a response. Common technologies, such as Atom or RSS feeds, are REST services.

As an alternative to using SOAP services, you can build Web services in PHP. You can use existing PHP and PEAR objects, such as SimpleXML, XMLReader, and XMLWriter included in PHP V5 (see Resources).

For troubleshooting steps, a good SOAP client that gives you all the SOAP faults helps. If your existing SOAP client swallows errors, consider using a different one. Other useful tools are those like Fiddler (see Resources) or other proxy applications that trap the messages sent to and from the Web service.

When I built the WSDL file for the first time, I wasn't paying attention and created the WSDL using **document literal**. While this is great practice for creating WSDL files, I had nothing but problems trying to get it to work, even with the PHP `SoapServer` class set to use `SOAP_DOCUMENT` and `SOAP_LITERAL`. The error that I received when the WSDL was set up as **document literal** was:

```
Procedure 'getRecommendedTireSpecs' not present.
```

After changing the WSDL to use **RPC encoded**, I got the error:

```
Error cannot find parameter.
```

I was able to make it go away be ridding the `automobileType` of all but one element (the year). I finally changed the WSDL to use **RPC literal**, which is fully Web Services Interoperability Organization (WS-I)-compliant, as well as allowing me to use the rich, complex types I had set up.

Section 8. Summary

Creating a WSDL file first, then the service, is doing contract-first development. With Eclipse, PDT, and WST, you have the tools you need to create the WSDL visually and easily, then use Eclipse's features to automatically publish your PHP service to a location.

The PDT project gives you the ability to write PHP code with first-class IDE features. Using the PHP perspective, you can navigate your PHP project more easily. With these tools and the built-in support in PHP for SOAP and XML, you can create PHP Web services with little effort.

Resources

Learn

- The online Eclipse help has more information about what's new, as well as additional tips and tricks.
- For more information about SOAP, see the PHP SOAP documentation.
- "XML for PHP developers" provides a great overview of using XML in PHP.
- PHP.net is the central resource for PHP developers.
- Check out the "Recommended PHP reading list."
- Browse all the PHP content on developerWorks.
- Expand your PHP skills by checking out IBM developerWorks' PHP project resources.
- Using a database with PHP? Check out the Zend Core for IBM, a seamless, out-of-the-box, easy-to-install PHP development and production environment that supports IBM DB2 V9.
- Check out the "Recommended Eclipse reading list."
- Browse all the Eclipse content on developerWorks.
- New to Eclipse? Read the developerWorks article "Get started with Eclipse Platform" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' Eclipse project resources.
- To listen to interesting interviews and discussions for software developers, check out developerWorks podcasts.
- Stay current with developerWorks' Technical events and webcasts.
- Watch and learn about IBM and open source technologies and product functions with the no-cost developerWorks On demand demos.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Download the Eclipse IDE and install it from the official site.

- Download the latest builds for the PDT project.
- The PHP site provides information on and functions for SimpleXML, XMLReader, and XMLWriter.
- Fiddler is the HTTP debugging proxy that works with most Internet browsers.
- Check out the latest Eclipse technology downloads at IBM alphaWorks.
- Download Eclipse Platform and other projects from the Eclipse Foundation.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The Eclipse newsgroups has many resources for people interested in using and extending Eclipse.
- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Nathan A. Good
Nathan Good lives in the Twin Cities area of Minnesota. When he isn't writing software, he enjoys building PCs and servers, reading about and working with new technologies, and trying to get all his friends to make the move to open source software. When he's not at a computer (which he admits isn't often), he spends time with his family, at his church, and at the movies. Visit his Web site.