



January | 12

Getting Started with Zephyr API for PHP

Revision History

Version	Type	Description/Comment	Date	Author
1.0	Creation	Initial release of the document.	7/01/11	Daniel Gannon

Table of Contents

1 - Overview	4
2 - Required Software.....	5
2.1 Installing PHP	5
2.2 Installing Eclipse PDT	6
2.2.1 Installing Zend Debugger	7
3 - Using Zephyr API with PHP	8
3.1 WSDL.....	8
3.2 Eclipse PDT IDE.....	8
3.2.1 Sample.php	9
3.2.2 Sample.php Output.....	12
3.3 Alternate Method.....	12
3.3.1 Sample.php	15
3.3.2 Sample.php Output.....	16

1 - Overview

How this guide will help

The “Getting Started with Zephyr API for PHP” guide will show you how to use Zephyr API within your PHP program, typically through the following steps:

- Utilizing the SOAP extension to expose the Zephyr API interfaces
- Writing scripts in PHP that exercise the Zephyr API

This guide assumes you already have a working knowledge of PHP. This guide will attempt to get you started by going over the software, in the “Required Software” section, which are necessary to perform the tasks detailed in the second half of the guide, “Using Zephyr API with PHP”. Versions of required software are subject to change, including Zephyr. It is recommended that you try to use the most up to date version of software listed, to ensure best compatibility with your system and security.

At the end of the document, sample code and output is provided to help get you started. After reading and understanding this guide, you will be able to understand and perform the basic operations, available to you through Zephyr’s API structure.

2 - Required Software



2.1 Installing PHP

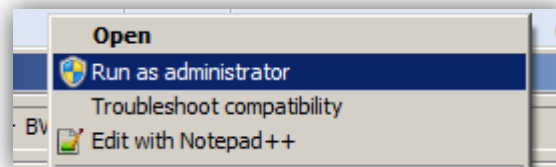
Before installing PHP or installing a newer version, make sure to uninstall all other unnecessary previous installations.

Download from: <http://www.php.net/downloads.php> - Source
<http://windows.php.net/download> - Windows Installer

Steps:

1. Run the PHP Installer

Navigate to the directory where you downloaded the executable. You must have administrative rights in order to install on Windows. This can be done by right-clicking the icon and then clicking "Run As Administrator" from the menu.



2. Enable Soap with PHP

In Windows installations, many extensions including SOAP come pre-enabled upon installation without the user having to input any commands.

In Linux or if you are compiling from source, the initial PHP setup and configuration process is controlled by the use of the command line options of the configure script. You could get a list of all available options along with short explanations. There are different options for each extension.

To enable SOAP support, configure and compile PHP with `--enable-soap`. This extension requires the libxml PHP extension. This means that passing in `--enable-libxml` is also required, although this is implicitly accomplished because libxml is enabled by default.

Further information on the SOAP extension can be found here:

<http://www.php.net/manual/en/book.soap.php>

3. Update the PHP_HOME variable (Optional)

You can run PHP without setting the PHP_HOME variable or you can set it for convenience. Setting the variables allows you to be able to conveniently run the executables from any directory without having to type the full command. Setting it this way will allow it to persist through reboot.

Setting up PHP_HOME in Windows (steps vary with Windows version):

1. Click **Start > Control Panel > System**
2. Click **Advanced > Environment Variables**
3. Scroll through the fields until you find PHP_HOME, if it's not present Add it to the System variables.
4. Add the location of the PHP root folder to the value of PHP_HOME in System variables (also User variables if present).

A typical path is: `C:\Program Files\PHP`

Hints

- Only one variable is allowed
- Only applies to new command windows opened after committing

4. Installation Complete

You should now be ready to use PHP code on your machine.

2.2 Installing Eclipse PDT

The PDT project provides a PHP Development Tools framework for the Eclipse platform. This project encompasses all development components necessary to develop PHP and facilitate extensibility. PDT will eventually become its own packaged product but currently works more like a plug-in to a current eclipse instance. Since Eclipse runs on a Java platform, you will need to install Java JDK (detailed below) before using Eclipse PDT.

Eclipse PDT IDE is just one answer when looking for an IDE to code with PHP. There are many more on the market, which may better suit your needs, however due to the popularity of Eclipse, this guide will attempt to show you how to use it when writing PHP code.

The recommended way to installing Eclipse PDT is by using the install new software tool in your current Eclipse instance, if applicable.

Steps:

1. In Eclipse, go to **Help>Install New Software**
2. A new window will come up. Input, <http://download.eclipse.org/releases/helios/> into the “Work With” field
3. Eclipse will parse the server input for Eclipse software
4. Browse down to the Programming Languages folder and expand
5. Check “PHP Development Tools (PDT) SDK Feature”
6. Click next and accept the ULA
7. Once you’ve finished downloading, restart Eclipse

For more information about this process, visit:

[http://wiki.eclipse.org/PDT/Installation#Eclipse 3.6 .2F Helios .2F PDT 2.2](http://wiki.eclipse.org/PDT/Installation#Eclipse_3.6_.2F_Helios_.2F_PDT_2.2)

You should now be able to run PHP code in Eclipse but you cannot yet debug code. For this you will need one of two possible PDT debuggers: XDeBug or Zend. See installation instructions below.

2.2.1 Installing Zend Debugger

When using Eclipse PDT, you have two options for debuggers; either Xend Debugger or XDeBug.

Links to both can be found at the PDT download site: <http://www.eclipse.org/pdt/downloads/>

For this guide, we will use the application debugger, Xend Debugger. Downloading Xend can be done in a way similar to installing Eclipse PDT.

Steps:

1. In Eclipse, go to **Help> Install New Software**
2. Input, <http://downloads.zend.com/pdt> into the “Work With” field
3. Check the plug-in and click Next
4. Accept the ULA
5. After installing, restart Eclipse

You should now be able to debug PHP code. Also, if you followed the guide in order, you should be able to write, run, and debug your code and are ready to start working with Zephyr API.

3 - Using Zephyr API with PHP

Reference URL of Zephyr API:

http://support.yourzephyr.com/api_help/

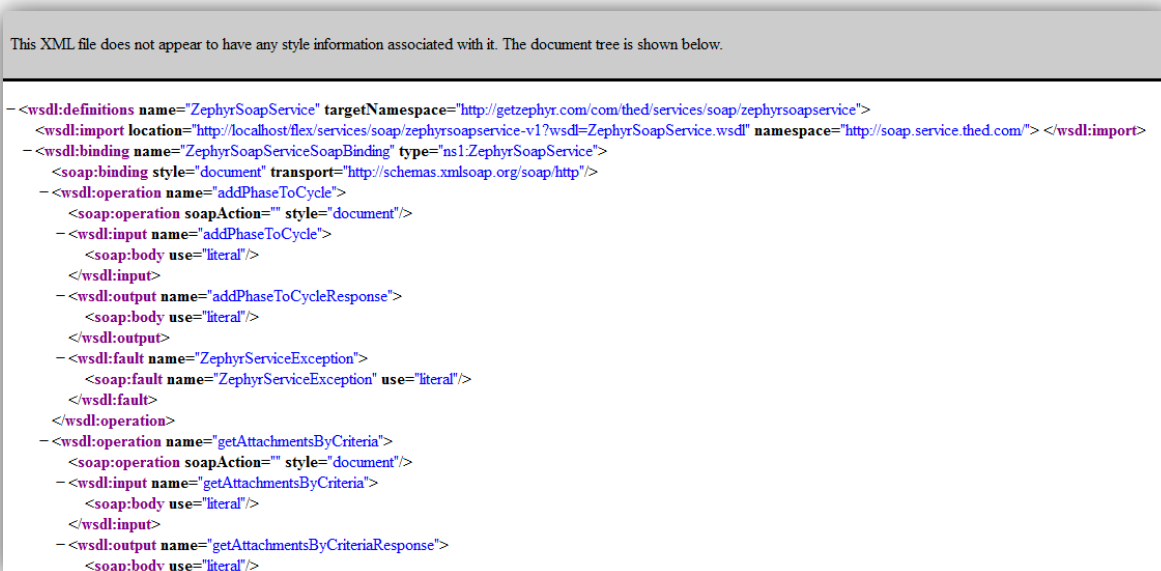
3.1 WSDL

The first step to using Zephyr API with PHP is to know and understand the WSDL for Zephyr. Understanding WSDL requires a basic knowledge of XML because it is an XML-based language for describing Web services and how to access them. The WSDL for your Zephyr service can be viewed by going to its URL from any browser, while the service is running, just change the URL to include your server name.

The URL for the WSDL is:

<http://<your zephyr server name>/flex/services/soap/zephyrsoapservice-v1?wsdl>

What you should see:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<wsdl:definitions name="ZephyrSoapService" targetNamespace="http://getzephyr.com/com/thed/services/soap/zephyrsoapservice">
  <wsdl:import location="http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl=ZephyrSoapService.wsdl" namespace="http://soap.service.thed.com"/> </wsdl:import>
  <soap:binding name="ZephyrSoapServiceSoapBinding" type="ns1:ZephyrSoapService">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  </wsdl:operation name="addPhaseToCycle">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="addPhaseToCycle">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="addPhaseToCycleResponse">
      <soap:body use="literal"/>
    </wsdl:output>
    <wsdl:fault name="ZephyrServiceException">
      <soap:fault name="ZephyrServiceException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
  <wsdl:operation name="getAttachmentsByCriteria">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="getAttachmentsByCriteria">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getAttachmentsByCriteriaResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:definitions>
```

3.2 Eclipse PDT IDE

As stated in the previous section, Eclipse PDT IDE is not the only answer when it comes to an IDE for PHP but is what will be used for this guide. If you are using another IDE or none at all, you may need to customize some of the steps to your environment.


```

*
////////////////////////////////////
////////
*/

// WSDL caching. 0(false) = off. 1(true) = on
ini_set('soap.wsdl_cache_enable',0);
ini_set('soap.wsdl_cache_ttl',0);

// Login class for setting login parameters
class login
{
    public $username;
    public $password;
}
// remoteCriteria class for setting searchCriteria parameters
class remoteCriteria
{
    public $searchName;
    public $searchOperation;
    public $searchValue;
}
// getProjectsByCriteria class for setting getProjectsByCriteria
parameters
class getProjectsByCriteria
{
    public $searchCriteria;
    public $returnAllDataFlag;
    public $token;
}

// URL for your WSDL file
$url = ("http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl");
// Options to customize your SoapClient
$options = array("exceptions"=> 1, 'soap_version'=>SOAP_1_1, 'trace'=> 1);

// Initialized login parameters
// username and password come from corresponding Zephyr user
$login_params = new login;
$login_params->username = "test.manager";
$login_params->password = "test.manager";

// Initialized remoteCriteria parameters
$sc = new remoteCriteria;
$sc->searchName = "id";
$sc->searchOperation = "EQUALS";
$sc->searchValue = "1";

// Initialized getProjectsByCriteria parameters
// searchCriteria is remoteCriteria initialized above

```

```

$pbcs_params = new getProjectsByCriteria;
$pbcs_params->searchCriteria = $sc;
$pbcs_params->returnAllDataFlag = "true";

// Create the SoapClient instance using $url variable
// Try/catch for fault detection
try {
    $client = new SoapClient($url, $options);
} catch (SoapFault $E) {
    echo "Exception Error!\n";
    echo $E->faultstring;
}
// echo for client confirmation
echo "Client Established. Running Login.\n";

// call to WSDL method login which takes in $login_params, resulting token
is saved in $session
$session = $client->login($login_params);

// echo for login confirmation
echo "The token for this session is: ";
echo ($session->return);
echo "\n\n";

// finished initialization of $pbcs_params from above, to add session token
$pbcs_params->token = $session->return ;

// call to WSDL method getProjectsByCriteria which takes in $pbcs_params,
returns all projects to $projects
$project = $client->getProjectsByCriteria($pbcs_params);

foreach ($project->return as $record)
{
    // echo for getProjectsByCriteria confirmation
    echo "The name of the project is: ";
    echo $record->name;
    echo "\n";
    echo "The start date of the project is: ";
    echo $record->startDate;
    echo "\n";
}
?>


```

3.2.2 Sample.php Output

```
Content-type: text/html
Client Established. Running Login.
The token for this session is: 535c90e6-c801-4ff8-89bf-428477895813

The name of the project is: Sample Project
The start date of the project is: 2010-12-20T00:00:00-08:00
The name of the project is: Project One
The start date of the project is: 2011-05-01T00:00:00-07:00
The name of the project is: NewPro
The start date of the project is: 2011-04-01T00:00:00-07:00
```

Please be aware that your project names and start dates (if you kept the sample intact) will likely be different from this sampled output. The idea behind giving the output is to allow you to look at the code and the output and connect what parts are doing what. Also, to check your output for comparison if need be.

With your code in place, be sure to save your work. After saving, make sure that no error symbols  in your code and then you can debug, compile, and run your code. With this example, debugging is not necessary but is a good practice to understand and use. Also, chances are, if you're using Eclipse IDE, your code is compiling automatically. A tell-tale sign of this is when you're writing code and error/ warning symbols appear without you clicking anything. This means that Eclipse has auto-compile enabled. Lastly, you will want to run your code to be sure that it performs expectedly and outputs correctly before possibly putting it into production. To run your code from the project explorer, **Select Class>Right-Click>Run-As>PHP Script**.

3.3 Alternate Method

There are many different ways to use PHP to code your application to work with the Zephyr API. The first method used just what is provided with a regular PHP install to use the Zephyr API. This alternate method will detail how to use a WSDL2PHP class convertor to generate a Zephyr API stubs class that you can use in your code.

For this alternate method, you need to download the latest version of WSDL2PHP class converter called wsdl2phpgenerator.

Download From: <https://github.com/walle/wsdl2phpgenerator>

Hint

- Download the latest after April 7th, 2011, as there were bugs in previous versions that prevented the correct generation of the stub classes

Be sure to uncompress the file in a convenient location for later use. Now, open a command prompt:

Windows 7/Vista

- Click on the start menu
- In the search bar, input: **CMD**
- Open CMD.EXE

Windows XP/NT/2000

- Click on the start menu
- Select Run
- Input: **CMD**
- Open CMD.EXE

Here are the parameters that the file will take:

Required Parameters	Description
-i	input The input file, must be a valid wsdl file
-o	output The output directory, is created if it does not exist and permission exists
Optional Parameters	
-e, --classExists, --exists	This flag tells the generator to surround all classes with if(!class_exists()) statements
-t, --noTypeConstructors	Tells the generator that no type constructor should be generated. Without this a standard constructor with all variables is generated.
-s, --singFile	Tells the generator to put all classes in the same file. The file is named after the service.
-n, --namespace	The name of the namespace to use if any. Requires php 5.3(or greater) on the server you are going to run this on.
-c	This flag should be a comma separated list of class names to generate. Used to only generate selected classes from the wsdl.
-p, --prefix	The prefix to use for the generated classes
-q, --suffix	The suffix to use for the generated classes
-h, --help, --h	Show the help section
--singleElementArrays	This flag tells the generator to insert the feature for single element arrays in the options array in the constructor of the service
--xsiArrayType	This flag tells the generator to insert the feature for the xsi array type in the options array in the constructor of the service
--waitOneWayCalls	This flag tells the generator to insert the feature for

	wait one way calls in the options array in the constructor of the service
--cacheNone	This flag tells the generator to insert the no cache flag in the options array in the constructor of the service
--cacheDisk	This flag tells the generator to insert the cache to disk flag in the options array in the constructor of the service
--cacheMemory	This flag tells the generator to insert the cache to memory flag in the options array in the constructor of the service
--cacheBoth	This flag tells the generator to insert the cache to disk and memory flag in the options array in the constructor of the service
--gzip	Enables gzip compression of the wsdl file

The command line structure for wsdl2phpgenerator use looks like:

```
cd %WSDL2PHPGENERATOR_HOME_DIRECTORY%
php wsdl2php [options] -i <WSDL_LOCATION> -o <OUT_FILE_LOCATION>
```

Here you can see that you will call the wsdl2php.php file in the directory. With that file select, you will call options and then give the input WSDL location. After that, you will give the location of where the output generation file will be placed. If a place does not exist at that location, the generator will create it.

I.E., for generation on a local installation:

```
cd C:\walle-wsdl2phpgenerator-b202ef8
php wsdl2php -s -v -l http://localhost/flex/services/soap/zephyrsoapservice-v1?wsdl -o C:\tempwsdl
```

```
C:\walle-wsdl2phpgenerator-b202ef8>php wsdl2php -s -v -l http://localhost/flex/
services/soap/zephyrsoapservice-v1?wsdl -o C:\temp
Starting generation
Loading the wsdl
Loading the DOM
Loading types
Done loading types
Starting to load service ZephyrSoapService
Loading function addPhaseIoCycle
Loading function getAttachmentsByCriteria
Loading function getProjectsByCriteria
Loading function login
Loading function createNewTestcaseTree
Loading function deleteAttachmentsByCriteria
Loading function updateIestcase
Loading function getIestcaseTreesByCriteria
Loading function getIestcaseTreeById
Loading function assignIestSchedules
Loading function getCustomFields
Loading function updateIestStatus
Loading function getIestSchedulesByCriteria
Loading function getIestcaseById
Loading function createNewCycle
Loading function getCyclesByCriteria
Loading function getProjectById
Loading function getReleaseById
Loading function createNewIestcase
Loading function getAttachmentById
Loading function getIestcasesByCriteria
Loading function getUserById
Loading function getUsersByCriteria
Loading function addAttachments
Loading function deleteAttachmentById
Loading function logout
Loading function getCycleById
Loading function getIestSchedulesById
Loading function updateAttachment
Loading function getReleasesByCriteria
Done loading service ZephyrSoapService
Generation complete
```



```

$login_param = new login("test.manager","test.manager");
$sc = new remoteCriteria("id", "EQUALS", "1");

// Create the SoapClient instance using $url variable
// Try/catch for fault detection
try {
    $client = new ZephyrSoapService($options, $wsdl);
} catch (SoapFault $E) {
    echo "Exception Error!<br>\n";
    echo $E->faultstring;
}
// echo for client confirmation
echo "Client Established. Running Login.\n";

// call to WSDL method login which takes in $login_params, resulting token
is saved in $session
$session = $client->login($login_param);

// echo for login confirmation
echo "The token for this session is: ";
echo ($session->return);
echo "\n\n";

// create the getProjectByCriteria using the searchCriteria, return flag,
and session token
$ppbc_param = new getProjectsByCriteria($sc, true, $session->return);

$projects = $client->getProjectsByCriteria($ppbc_param);

foreach ($projects->return as $record)
{
    // echo for getProjectsByCriteria confirmation
    echo "The name of the project is: ";
    echo $record->name;
    echo "\n";
    echo "The start date of the project is: ";
    echo $record->startDate;
    echo "\n";
}
?>

```

3.3.2 Sample.php Output

```


Content-type: text/html
Client Established. Running Login.
The token for this session is: 56bcbe9b-63e4-47e8-ab2e-d84e12dcc161

```

```
The name of the project is: Sample Project
The start date of the project is: 2010-12-20T00:00:00-08:00
The name of the project is: Project One
The start date of the project is: 2011-05-01T00:00:00-07:00
The name of the project is: NewPro
The start date of the project is: 2011-04-01T00:00:00-07:00
```

As you can see, despite taking two different methods to write the code, both methods have the same output. On an evaluation basis, the Sample.php from the second method would be recommended for newer PHP users who are not as familiar with PHP and how it works. Also the second method will allow you to produce code quicker over a longer period due to needing to write fewer lines to perform the same actions.

Please be aware that your project names and start dates (if you kept the sample intact) will likely be different from this sampled output. The idea behind giving the output is to allow you to look at the code and the output and connect what parts are doing what. Also, to check your output for comparison if need be.

With your code in place, be sure to save your work. After saving, make sure that no error symbols  in your code and then you can debug, compile, and run your code. With this example, debugging is not necessary but is a good practice to understand and use. Also, chances are, if you're using Eclipse IDE, your code is compiling automatically. A tell-tale sign of this is when you're writing code and error/ warning symbols appear without you clicking anything. This means that Eclipse has auto-compile enabled. Lastly, you will want to run your code to be sure that it performs expectedly and outputs correctly before possibly putting it into production. To run your code from the project explorer, **Select Class>Right-Click>Run-As>PHP Script.**