Using Amazon Web Services with PHP & SOAP part one

By icarus

Table of Contents

A To Z, And Everything In Between

Everyone, but everyone, knows what Amazon.com is − it's the largest (and, for my money, coolest) online store on the planet, selling everything from baby clothes to the new Volkswagen Beetle. It's been around since the beginning of the Web and offers one of the friendliest shopping experiences online, together with great customer service and a wide variety of discounts.

One of the reasons for Amazon's dominance in the online shopping space is its creativity − the store's managers are constantly coming up with innovative new ideas to simplify and enhance the customer experience. And one of the cooler new ideas to emerge from Amazon HQ in recent months has been Amazon Web Services, a set of APIs designed to let users query the complete Amazon database using a series of SOAP−based remote procedure calls. These Web services allow regular users to easily create online stores that leverage off Amazon's experience (and huge product catalog), and to build cutting−edge e−commerce applications quickly and efficiently.

Now, your favourite language and mine, PHP, has recently started shipping with support for XML−based remote procedure calls (including SOAP) over HTTP. This makes PHP ideal for developers looking to integrate Amazon Web Services into their Web applications. The only problem? Not too many people know how to do it.

That's where this tutorial comes in. Over the next few pages, I'll be demonstrating how you can use PHP, in combination with Amazon Web Services, to add powerful new capabilities to your Web applications. Take a look.

Remote Control

Before we get into the code, though, you need to have a clear understanding of how Amazon Web Services, aka AWS, works. This involves getting up close and personal with a complicated little critter known as SOAP, the Simple Object Access Protocol.

According to the primer available on the W3C's site (http://www.w3.org/2002/ws/), SOAP is a "lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol at the core of which is an envelope that defines a framework for describing what is in a message and how to process it and a transport binding framework for exchanging messages using an underlying protocol."

If you're anything like me, that probably made very little sense to you. So here's the Reader's Digest version, which is far more cogent: "SOAP is a simple XML based protocol to let applications exchange information over HTTP." (http://www.w3schools.com/soap/soap_intro.asp)

SOAP is a client−server paradigm which builds on existing Internet technologies to simplify the task of invoking procedures and accessing objects across a network. It uses XML to encode procedure invocations (and decode procedure responses) into a package suitable for transmission across a network, and HTTP to actually perform the transmission.

At one end of the connection, a SOAP server receives SOAP requests containing procedure calls, decodes them, invokes the function and packages the response into a SOAP packet suitable for retransmission back to the requesting client. The client can then decode the response and use the results of the procedure invocation in whatever manner it chooses. The entire process is fairly streamlined and, because of its reliance on existing standards, relatively easy to understand and use.

Here's a quick example of what a SOAP request for the procedure getFlavourOfTheDay() might look like:

```
<?xml version="1.0" ?>
<SOAP−ENV:Envelope
xmlns:SOAP−ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xmlns:SOAP−ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:ns6="http://testuri.org"
SOAP−ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP−ENV:Body>
<ns6:getFlavourOfTheDay>
<day xsi:type="xsd:string">monday</day>
</ns6:getFlavourOfTheDay>
</SOAP−ENV:Body>
</SOAP−ENV:Envelope>
```

And here's what the response might look like:

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:ns6="http://testuri.org"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns6:getFlavourOfTheDayResponse>
<flavour xsi:type="xsd:string">pineapple</flavour>
</ns6:getFlavourOfTheDayResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

I'm not going to get into the details of how SOAP works in this article, preferring instead to focus on how SOAP can be exploited in the context of PHP and AWS. If you're new to SOAP, the information above should be sufficient to explain the basic concepts and ensure that you can follow the material that comes next; however, if you're interested in learning more about SOAP (or if you just have trouble falling asleep at night), you should read the W3C's specifications on the protocol − links will be included at the end of this article.

The Bare Necessities

here are a couple of things you'll need before you can get started with PHP and AWS. Obviously, you need a working build of PHP − I recommend the latest version, PHP 4.2.3, which you can download from http://www.php.net/

You'll also need an external PHP class named NuSOAP, which exposes a number of methods that can be used to instantiate a SOAP client and perform SOAP transactions over HTTP. You can download NuSOAP from http://dietrich.ganx4.com/.

Finally, you need a ticket to the Amazon.com gravy train. Drop by http://www.amazon.com/webservices/, register with Amazon.com, and pick up your free developer token. This developer token will be used in all your interaction with AWS, so handle it carefully − you're going to need it very soon.

While you're on the Amazon.com Web site, you might also want to download the AWS software development kit, which contains numerous examples of how AWS can be used on different platforms, together with detailed documentation of the AWS API. Be sure to read the SOAP development guidelines in the AWS documentation, so that you don't inadvertently burn down Amazon's servers.

All set up? Let's rock and roll.

Anatomy Class

Amazon has made a number of important method calls available in the AWS API − here's a brief list:

BrowseNodeSearchRequest() − retrieve a list of catalog items attached to a particular node in the Amazon database;

ASINSearchRequest() − retrieve detailed information for a given product code;

KeywordSearchRequest() − perform a keyword search on the Amazon database;

SellerSearchRequest() − perform a search for products listed by third−party sellers;

PowerSearchRequest() − perform an advanced search on the Amazon database;

SimilaritySearchRequest() − perform a search for similar items, given a specific product code.

Additionally, AWS includes methods to search for authors, actors, directors, artists, manufacturers, wish lists and user lists.

This might not seem like much to start with − but, as you'll see, looks are deceptive. Consider the following simple example, which provides a gentle introduction to the power of AWS:

```
<?php

// include class
include("nusoap.php");

// create a instance of the SOAP client object
// remember that this script is the client,
// accessing the web service provided by Amazon.com
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient−>debug_flag = 1;

// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient−>getProxy();

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'lite',
'devtag' => 'YOUR-TOKEN-HERE'
);


// invoke the method
$result = $proxy->BrowseNodeSearchRequest($params);

// print the results of the search
print_r($result);
?>
```

The first order of business is to include the SOAP class which contains all the methods needed to access SOAP services.

```
// include the class
include("nusoap.php");
?>
```

Now, in this SOAP universe, Amazon provides the SOAP server, and this PHP script works as the client. So, the next step is to instantiate this client, using the class constructs provided by NuSOAP.

```
<?php
// create a instance of the SOAP client object
// remember that this script is the client,
// accessing the web service provided by Amazon.com
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true); ?>
```

The class constructor accepts a single parameter, which is the URL of the SOAP service to be accessed (this is sometimes referred to by geeks as the "endpoint"). In case you're wondering where I got this URL from − it's listed in the AWS documentation. So there!

In order to simplify usage of the AWS API, I've created a "proxy client", one which lets me directly invoke AWS methods, rather than passing them to the NuSOAP class' call() method.

```
<?php
// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient->getProxy(); ?>
```

All that remains is to send a request to the SOAP server via this proxy,

```
<?php
// invoke the method
$result = $proxy->BrowseNodeSearchRequest($params);
?>
```

and print the resulting output.

```
<?php
// print the results of the search
print_r($result);
?>
```

In this case, I'm calling the BrowseNodeSearchRequest() method on the AWS SOAP server, and passing it a list of arguments (this argument list is also documented in the AWS API). This argument list is stored in the $params array, which looks like this:

```
<?php
// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'lite',
'devtag' => 'YOUR-TOKEN-HERE'
);
?>
```

Here's what those arguments mean:

1. The "browse_node" argument specifies the node to begin with in the catalog. This node ID can be obtained by visiting the Amazon.com Web site and looking at the URL for the section you're interested in browsing. For example, the URL

```
http://www.amazon.com/exec/obidos/tg/browse/-/18
```

points to the Mystery category in Amazon's bookstore and has node ID 18.

In case this seems like too much work, a list of the most popular node IDs is available in the AWS documentation.

2. The "page" argument specifies the page offset to display. AWS is currently hard−wired to display 10 items per page, so if you wanted to display items 11−20, you would need to set

```
'page' => 2
```

and so on.

3. The "mode" argument specifies the particular store to browse. As of this writing, AWS defines 16 stores, each with a unique "mode" identifier − here's a list of the ones I visit most often:

```
Books:
'mode' => 'books' Popular Music:
'mode' => 'music'
Electronics:
'mode' => 'electronics' DVD:
'mode' => 'dvd' Computers:
'mode' => 'pc−hardware' Software:
'mode' => 'software'
Toys:
'mode' => 'toys'
```

Again, a complete list of stores is available in the AWS documentation.

4. The "tag" argument specifies your Amazon.com Associates ID, if you have one. In case you don't, and if you're serious about building an online store with AWS, I suggest you get one post−haste from http://associates.amazon.com/, since it entitles you to a commission on every purchase made via your site.

5. The "type" argument specifies the type of result set you would like. AWS gives you two choices − "lite", which contains basic product information, and "heavy", which contains detailed product information. I'll show you both in this article.

6. Finally, remember that developer token you got when you first registered for AWS? You need to specify it via the "devtag" argument in order to use AWS; if it isn't included in the argument list, AWS will deny you access.

If you take a look at the internals of the SOAP class, you'll see that the proxy uses the class' call() method and the arguments passed to it to generate a SOAP request, which looks something like this:

```
POST /onca/soap2 HTTP/1.0
User-Agent: NuSOAP/0.6.3
Host: soap.amazon.com
Content-Type: text/xml
Content-Length: 942
SOAPAction: "urn:PI/DevCentral/SoapService"

<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd"
xmlns:typens="urn:PI/DevCentral/SoapService">
<SOAP-ENV:Body><typens:BrowseNodeSearchRequest>
<BrowseNodeSearchRequest xsi:type="typens:BrowseNodeRequest">
<browse_node xsi:type="xsd:string">18</browse_node><page
xsi:type="xsd:string">1</page><mode
xsi:type="xsd:string">books</mode><tag
xsi:type="xsd:string">melonfire-20</tag><type
xsi:type="xsd:string">lite</type><devtag
xsi:type="xsd:string">YOUR-TOKEN-HERE</devtag><sort
xsi:type="xsd:string">18</sort></BrowseNodeSearchRequest></typens:Browse>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This request packet is transmitted to the SOAP server using the POST method, and a server response packet is transmitted back to the client. Here's what one such packet might look like:

```
HTTP/1.1 200 OK
Date: Tue, 05 Nov 2002 15:38:54 GMT
Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Unix)
mod_fastcgi/2.2.10
Connection: close
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:amazon="urn:PI/DevCentral/SoapService">
```

```
<SOAP−ENV:Body>
<namesp44:BrowseNodeSearchRequestResponse
xmlns:namesp44="urn:PI/DevCentral/SoapService">

<return xsi:type="amazon:ProductInfo">
<TotalResults xsi:type="xsd:string">55656</TotalResults>
<Details SOAP−ENC:arrayType="amazon:Details[10]"
xsi:type="SOAP−ENC:Array"> <Details
xsi:type="amazon:Details"><Url
xsi:type="xsd:string">http://www.amazon.com/exec/obidos/redirect?tag=mel
onfi
re−20%26creative=YOUR−TOKEN−HERE%26camp=2025%26link_code=sp1%26path=ASIN
/006
0092572</Url><Asin
xsi:type="xsd:string">0060092572</Asin><ProductName
xsi:type="xsd:string">The Terminal Man</ProductName><Catalog
xsi:type="xsd:string">Book</Catalog><Authors
SOAP−ENC:arrayType="xsd:string[1]"
xsi:type="SOAP−ENC:Array"><Author
xsi:type="xsd:string">Michael
Crichton</Author></Authors><ReleaseDate
xsi:type="xsd:string">05 November,
2002</ReleaseDate><Manufacturer
xsi:type="xsd:string">Avon</Manufacturer><ImageUrlSmall
xsi:type="xsd:string">http://images.amazon.com/images/P/0060092572.01.TH
UMBZ
ZZ.jpg</ImageUrlSmall><ImageUrlMedium
xsi:type="xsd:string">http://images.amazon.com/images/P/0060092572.01.MZ
ZZZZ
ZZ.jpg</ImageUrlMedium><ImageUrlLarge
xsi:type="xsd:string">http://images.amazon.com/images/P/0060092572.01.LZ
ZZZZ
ZZ.jpg</ImageUrlLarge><ListPrice
xsi:type="xsd:string">$7.99</ListPrice><OurPrice
xsi:type="xsd:string">$7.99</OurPrice><UsedPrice
xsi:type="xsd:string">$4.00</UsedPrice>

</Details>
<Details xsi:type="amazon:Details"><Url
xsi:type="xsd:string">http://www.amazon.com/exec/obidos/redirect?tag=mel
onfi
re−20%26creative=YOUR−TOKEN−HERE%26camp=2025%26link_code=sp1%26path=ASIN
/006
0505397</Url><Asin
xsi:type="xsd:string">0060505397</Asin><ProductName
xsi:type="xsd:string">Bet Your Life</ProductName><Catalog
xsi:type="xsd:string">Book</Catalog><Authors
SOAP−ENC:arrayType="xsd:string[1]"
```

```
xsi:type="SOAP-ENC:Array"><Author
xsi:type="xsd:string">Richard
Dooling</Author></Authors><ReleaseDate
xsi:type="xsd:string">05 November,
2002</ReleaseDate><Manufacturer
xsi:type="xsd:string">HarperCollins</Manufacturer><ImageUrlSmall
xsi:type="xsd:string">http://images.amazon.com/images/P/0060505397.01.TH
UMBZ
ZZ.jpg</ImageUrlSmall><ImageUrlMedium
xsi:type="xsd:string">http://images.amazon.com/images/P/0060505397.01.MZ
ZZZZ
ZZ.jpg</ImageUrlMedium><ImageUrlLarge
xsi:type="xsd:string">http://images.amazon.com/images/P/0060505397.01.LZ
ZZZZ
ZZ.jpg</ImageUrlLarge><ListPrice
xsi:type="xsd:string">$25.95</ListPrice><OurPrice
xsi:type="xsd:string">$18.17</OurPrice><UsedPrice
xsi:type="xsd:string">$12.98</UsedPrice>

</Details>

... and so on...

</Details>
</return>
</namesp44:BrowseNodeSearchRequestResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This response packet is decoded by the client into a native PHP structure, which can be used within the script. At the moment, all I'm doing is printing it − and here's what the output looks like:

```
Array
(
[TotalResults] => 55656
[Details] => Array
(
[0] => Array
(
[Url] =>

http://www.amazon.com/exec/obidos/redirect?tag=melonfire-20%26creative=Y
OUR-
TOKEN-HERE%26camp=2025%26link_code=sp1%26path=ASIN/0060092572
[Asin] => 60092572
[ProductName] => The Terminal Man
[Catalog] => Book
[Authors] => Array
(
[0] => Michael Crichton
)

[ReleaseDate] => 05 November, 2002
[Manufacturer] => Avon
[ImageUrlSmall] =>
http://images.amazon.com/images/P/0060092572.01.THUMBZZZ.jpg
[ImageUrlMedium] =>

http://images.amazon.com/images/P/0060092572.01.MZZZZZZZ.jpg
[ImageUrlLarge] =>
http://images.amazon.com/images/P/0060092572.01.LZZZZZZZ.jpg
[ListPrice] => $7.99
[OurPrice] => $7.99
[UsedPrice] => $4.00
)

[1] => Array
(
[Url] =>
http://www.amazon.com/exec/obidos/redirect?tag=melonfire−20%26creative=Y
OUR−
TOKEN−HERE%26camp=2025%26link_code=sp1%26path=ASIN/0060505397
[Asin] => 60505397
[ProductName] => Bet Your Life
[Catalog] => Book
[Authors] => Array
(
[0] => Richard Dooling
)

[ReleaseDate] => 05 November, 2002
[Manufacturer] => HarperCollins
[ImageUrlSmall] =>

http://images.amazon.com/images/P/0060505397.01.THUMBZZZ.jpg
[ImageUrlMedium] =>
http://images.amazon.com/images/P/0060505397.01.MZZZZZZZ.jpg
[ImageUrlLarge] =>
http://images.amazon.com/images/P/0060505397.01.LZZZZZZZ.jpg
[ListPrice] => $25.95
[OurPrice] => $18.17
[UsedPrice] => $12.98
)

... and so on ...
```

```
[9] => Array
(
[Url] =>
http://www.amazon.com/exec/obidos/redirect?tag=melonfire-20%26creative=Y
OUR-
TOKEN-HERE%26camp=2025%26link_code=sp1%26path=ASIN/0195122623
[Asin] => 195122623
[ProductName] => Arthur Conan Doyle: Beyond Baker
Street (Oxford Portraits Series)
[Catalog] => Book
[Authors] => Array
(
[0] => Janet B. Pascal
)

[ReleaseDate] => March, 2000
[Manufacturer] => Oxford Univ Pr Childrens Books
[ImageUrlSmall] =>

http://images.amazon.com/images/P/0195122623.01.THUMBZZZ.jpg
[ImageUrlMedium] =>
http://images.amazon.com/images/P/0195122623.01.MZZZZZZZ.jpg
[ImageUrlLarge] =>
http://images.amazon.com/images/P/0195122623.01.LZZZZZZZ.jpg
[ListPrice] => $24.00
[OurPrice] => $24.00
[UsedPrice] => $5.49
)

)

)
```

Obviously, this is not very useful − but we're just getting started. Flip the page, and I'll show you how to massage all this raw data into something a little less ugly.

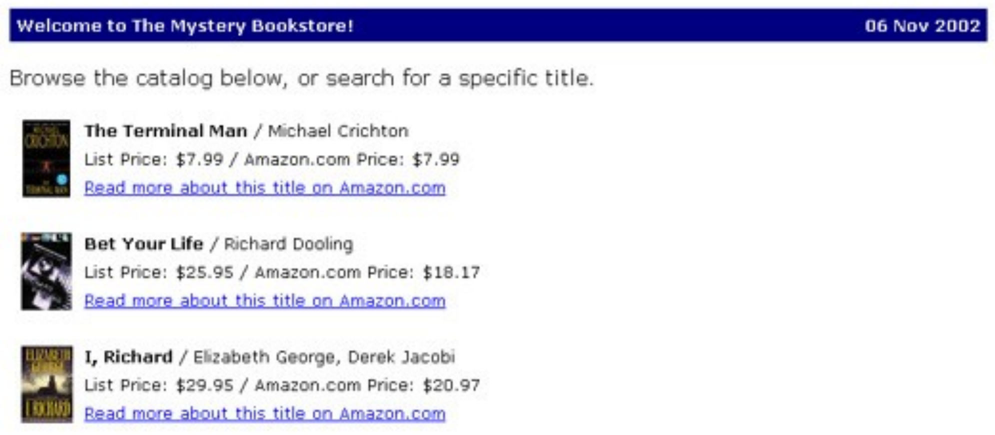
The Bookworm Turns

If you take a close look at the output of the previous example, you'll see that the call to BrowseNodeSearchRequest() results in a PHP associative array containing a series of result elements. It's extremely simple to convert the raw data contained within this array into a properly−formatted HTML page. Watch!

```
<?php
// include class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient->getProxy();

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire−20',
'type' => 'lite',
'devtag' => 'YOUR−TOKEN−HERE'
);

// invoke the method
$result = $proxy−>BrowseNodeSearchRequest($params);

if ($result['faultstring'])
{
?>
<h2>Error</h2>
<? echo $result['faultstring'];?>
<?
}
else
{
$total = $result['TotalResults'];
```

```
$items = $result['Details'];

// format and display the results
?>

<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">

<p>&nbsp;<p>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td bgcolor="Navy"><font color="white" size="−1"><b>Welcome to
The Mystery Bookstore!</b></font></td>
<td bgcolor="Navy" align="right"><font color="white"
size="−1"><b><? echo date("d M Y",
mktime());?></b></font></td> </tr>
</table>

<p>

Browse the catalog below, or search for a specific title.
<p>

<table width="100%" border="0" cellspacing="5"
cellpadding="0"> <?

// parse the $items[] array and extract the necessary
information
// (image, price, title, author, item URL)
foreach ($items as $i)
{
?>
<tr>
<td align="center" valign="top" rowspan="3"><a href="<? echo
$i['Url'];
?>"><img border="0" src=<? echo $i['ImageUrlSmall'];
?>></a></td>
<td><font size="−1"><b><? echo $i['ProductName']; ?></b> / <?
echo
implode(", ", $i['Authors']); ?></font></td> </tr> <tr> <td
align="left"
valign="top"><font size="−1">List Price: <? echo
$i['ListPrice']; ?> /
Amazon.com Price: <? echo $i['OurPrice']; ?></font></td> </tr>
<tr> <td
align="left" valign="top"><font size="−1"><a href="<? echo
$i['Url'];
?>">Read more about this title on Amazon.com</a></font></td>
</tr> <tr>
<td colspan=2>&nbsp;</td> </tr> <? } ?> </table>

<font size="−1">
Disclaimer: All product data on this page belongs to
Amazon.com. No
guarantees are made as to accuracy of prices and information.
YMMV!
</font>

</body>
</html>
<?
}
?>
```

Here's what it looks like:



Welcome to The Mystery Bookstore! 06 Nov 2002

Browse the catalog below, or search for a specific title.

The Terminal Man / Michael Crichton
List Price: $7.99 / Amazon.com Price: $7.99
Read more about this title on Amazon.com

Bet Your Life / Richard Dooling
List Price: $25.95 / Amazon.com Price: $18.17
Read more about this title on Amazon.com

I, Richard / Elizabeth George, Derek Jacobi
List Price: $29.95 / Amazon.com Price: $20.97
Read more about this title on Amazon.com

There's no magic here − all I've done is taken the associative array created by NuSOAP from the SOAP response and massaged its elements into a properly−laid out Web page. Most of the work happens in the "foreach" loop, which iterates through the result array and displays the items in a table, complete with thumbnail images.

In case you're wondering where all the data came from, flip back to the previous page and look at the keys of the associative array generated by NuSOAP − you'll see that the name of each key is self−explanatory, and maps into the data displayed on the page above. Notice also that each item is accompanied by a link to the product information page on the actual Amazon.com Web site, and that this URL (obtained from the "Url" key of the result array) also contains an embedded Associates ID so that Amazon can send you a commission in case the click results in an actual sale.

In the event that the procedure generates an error on the server, the response array will contain a SOAP fault. It's generally considered good programming practice to check for this and handle it appropriately − you'll see that I've done this in the script above.

Sorting Things Out

Now, the list displayed on the previous page is sorted in the default order imposed by Amazon.com. However, AWS allow you to alter this sort order by specifying an optional "sort" argument in the call to BrowseNodeSearchRequest(). This "sort" argument allows you to sort products by price, by sales rank, by rating, by date or alphabetically.

In order to demonstrate this, consider the following enhancement to the example on the previous page, which performs three BrowseNodeSearchRequest() calls, each one applying a different sort criteria. The first one displays items in the default order; the second displays featured items first; and the third displays items by sales rank. Notice how the results of these three AWS calls can be massaged to create a more dynamic, informative and user−friendly page.

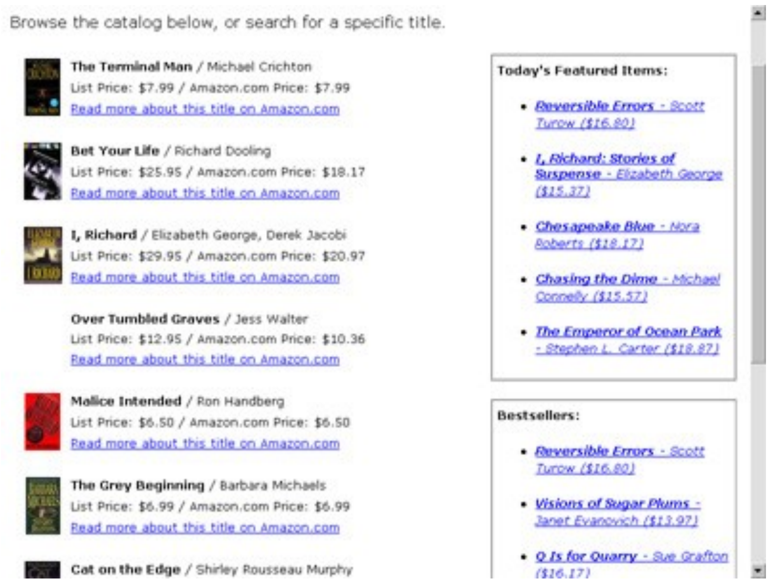
```
<?php
// include class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient->getProxy();

// get items from the catalog
// sort order is default
$catalogParams = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire−20',
'type' => 'lite',
'devtag' => 'YOUR−TOKEN−HERE'
);
$catalogResult =
$proxy->BrowseNodeSearchRequest($catalogParams);
$catalogTotal = $catalogResult['TotalResults'];
$catalogItems = $catalogResult['Details'];

// get today's featured items
// sort order is by featured items
$featuredParams = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'lite',
'sort' => '+pmrank',
'devtag' => 'YOUR-TOKEN-HERE'
);
$featuredResult =
$proxy->BrowseNodeSearchRequest($featuredParams);
$featuredTotal = $featuredResult['TotalResults'];
$featuredItems =
$featuredResult['Details'];

// get bestsellers
// sort order is by sales ranking
$bestsellersParams = array(
'browse_node' => 18,
'page' => 1,
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'lite',
'sort' => '+salesrank',
'devtag' => 'YOUR-TOKEN-HERE'
);
$bestsellersResult =
$proxy->BrowseNodeSearchRequest($bestsellersParams);
$bestsellersTotal = $bestsellersResult['TotalResults'];
$bestsellersItems = $bestsellersResult['Details'];

// format and display the results
?>

<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">

<p>&nbsp;<p>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td bgcolor="Navy"><font color="white" size="-1"><b>Welcome to
The Mystery Bookstore!</b></font></td>
<td bgcolor="Navy" align="right"><font color="white"
size="-1"><b><? echo date("d M Y",
mktime());?></b></font></td> </tr>
```

```
</table>

<p>

Browse the catalog below, or search for a specific title.
<p>

<!-- outer table -->
<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td align="left" valign="top" rowspan="2" width="65%">
<!-- inner catalog table -->

<table border="0" cellspacing="5" cellpadding="0">
<?

// parse the $items[] array and extract the necessary
information
// (image, price, title, author, item URL)
foreach ($catalogItems as $i)
{
?>
<tr>
<td align="center" valign="top" rowspan="3"><a href="<?
echo $i['Url']; ?>"><img border="0" src=<? echo
$i['ImageUrlSmall'];
?>></a></td>
<td><font size="-1"><b><? echo $i['ProductName']; ?></b>
/ <? echo implode(", ", $i['Authors']); ?></font></td>
</tr>
<tr>
<td align="left" valign="top"><font size="-1">List
Price: <? echo $i['ListPrice']; ?> / Amazon.com Price: <? echo
$i['OurPrice']; ?></font></td>
</tr>
<tr>
<td align="left" valign="top"><font size="-1"><a
href="<? echo $i['Url']; ?>">Read more about this title on
Amazon.com</a></font></td>
</tr>
<tr>
<td colspan=2>&nbsp;</td>
</tr>
<?
}
?>
</table>

</td>
```

```
<td valign="top">

<font size="-1">
<!-- featured item table -->
<table border="1" cellspacing="0" cellpadding="5">
<tr>
<td>
<b><font size="-1">Today's Featured Items:</font></b>
<ul>
<?
for ($x=0; $x<5; $x++)
{
$f = $featuredItems[$x];
?>
<li><i><a href="<? echo $f['Url']; ?>"><font
size="-1"><b><? echo $f['ProductName']; ?></b> - <? echo
implode(", ",
$f['Authors']); ?> (<? echo $f['OurPrice']; ?>)</font></a></i>
<p>
<?
}
?>
</ul>
</td>
</tr>
</table>

<p>

<!-- bestseller list -->
<table border="1" cellspacing="0" cellpadding="5">
<tr>
<td>
<b><font size="-1">Bestsellers:</font></b>
<ul>
<?
for ($y=0; $y<5; $y++)
{
$b = $bestsellersItems[$y];
?>
<li><i><a href="<? echo $b['Url']; ?>"><font
size="-1"><b><? echo $b['ProductName']; ?></b> - <? echo
implode(", ",
$b['Authors']); ?> (<? echo $b['OurPrice']; ?>)</font></a></i>
<p>
<?
}
?>
</ul>
```

```
</td>
</tr>
</table>
</font>
</td>
</tr>
</table>

<font size="-1">
Disclaimer: All product data on this page belongs to
Amazon.com. No
guarantees are made as to accuracy of prices and information.
YMMV!
</font>

</body>
</html>
```

In this case, the additional "sort" argument is used to obtain a list of featured items and bestsellers within the Mystery node of the Amazon book database. Here's what the output looks like:



Browse the catalog below, or search for a specific title.

The Terminal Man / Michael Crichton
List Price: $7.99 / Amazon.com Price: $7.99
Read more about this title on Amazon.com

Bet Your Life / Richard Dooling
List Price: $25.95 / Amazon.com Price: $18.17
Read more about this title on Amazon.com

I, Richard / Elizabeth George, Derek Jacobi
List Price: $29.95 / Amazon.com Price: $20.97
Read more about this title on Amazon.com

Over Tumbled Graves / Jess Walter
List Price: $12.95 / Amazon.com Price: $10.36
Read more about this title on Amazon.com

Malice Intended / Ron Handberg
List Price: $6.50 / Amazon.com Price: $6.50
Read more about this title on Amazon.com

The Grey Beginning / Barbara Michaels
List Price: $6.99 / Amazon.com Price: $6.99
Read more about this title on Amazon.com

Cat on the Edge / Shirley Rousseau Murphy

Today's Featured Items:

- *Reversible Errors* - Scott Turow ($16.80)
- *I, Richard: Stories of Suspense* - Elizabeth George ($15.37)
- *Chesapeake Blue* - Nora Roberts ($18.17)
- *Chasing the Dime* - Michael Connelly ($15.57)
- *The Emperor of Ocean Park* - Stephen L. Carter ($18.87)

Bestsellers:

- *Reversible Errors* - Scott Turow ($16.80)
- *Visions of Sugar Plums* - Janet Evanovich ($13.97)
- *Q Is for Quarry* - Sue Grafton ($16.17)

A number of other sort criteria are available in AWS − take a look at the documentation for details.

Turning The Pages

You'll remember, from my explanation of the various arguments to BrowseNodeSearchRequest() a few pages back, that AWS returns search results in chunks of ten, and the "page" argument must be used to obtain subsequent pages of the result set.

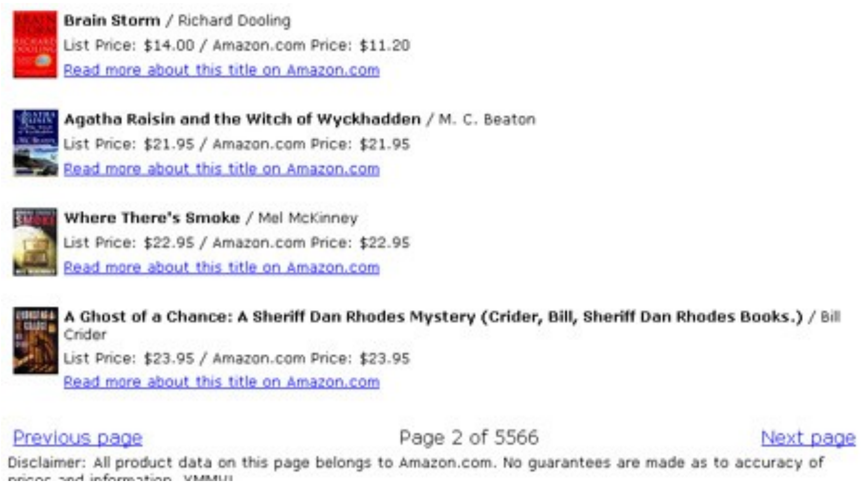
Thus far, all the examples you've seen have been limited to displaying ten items...not very useful in the real world at all. That's why this next example adds previous and next page links to assist in navigating between the different pages of the result set.

```
<?php

// include class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// if no page specified, start with page 1
if (!$_GET['page']) { $page = 1; } else { $page =
$_GET['page']; }

// create a proxy so that WSDL methods can be accessed
directly $proxy =
$soapclient->getProxy();

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'browse_node' => 18,
'page' => $page,
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'lite',
'devtag' => 'YOUR-TOKEN-HERE'
);

// invoke the method
$result = $proxy->BrowseNodeSearchRequest($params);

$total = $result['TotalResults'];
$items = $result['Details'];
```

```
// format and display the results
?>

<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">

<p>&nbsp;<p>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td bgcolor="Navy"><font color="white" size="−1"><b>Welcome to
The Mystery Bookstore!</b></font></td>
<td bgcolor="Navy" align="right"><font color="white"
size="−1"><b><? echo date("d M Y",
mktime());?></b></font></td> </tr>
</table>

<p>

Browse the catalog below, or search for a specific title.
<p>

<table width="100%" border="0" cellspacing="5"
cellpadding="0"> <?

// parse the $items[] array and extract the necessary
information
// (image, price, title, author, item URL)
foreach ($items as $i)
{
?>
<tr>
<td align="center" valign="top" rowspan="3"><a href="<? echo
$i['Url'];
?>"><img border="0" src=<? echo $i['ImageUrlSmall'];
?>></a></td>
<td><font size="−1"><b><? echo $i['ProductName']; ?></b> / <?
echo
implode(", ", $i['Authors']); ?></font></td> </tr> <tr> <td
align="left"
valign="top"><font size="−1">List Price: <? echo
$i['ListPrice']; ?> /
Amazon.com Price: <? echo $i['OurPrice']; ?></font></td> </tr>
<tr> <td
align="left" valign="top"><font size="-1"><a href="<? echo
$i['Url'];
?>">Read more about this title on Amazon.com</a></font></td>
</tr> <tr>
<td colspan=2>&nbsp;</td> </tr> <? } ?> </table>

<!-- next and prev page links -->

<?
$pageCount = ceil($total/10);
?>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td align="left">
<?
if ($page != 1)
{
?>
<a href="<? echo $_SERVER['PHP_SELF']; ?>?page=<? echo
$page−1;
?>">Previous page</a> <? } ?> &nbsp; </td> <td
align="center">Page <?
echo $page; ?> of <? echo $pageCount; ?></td> <td
align="right"> &nbsp;
<? if ($page < $pageCount) { ?> <a href="<? echo
$_SERVER['PHP_SELF'];
?>?page=<? echo $page+1; ?>">Next page</a> <? } ?> </td> </tr>
</table>

<font size="−1">
Disclaimer: All product data on this page belongs to
Amazon.com. No
guarantees are made as to accuracy of prices and information.
YMMV!
</font>

</body>
</html>
```

How does this work? It's actually pretty simple − first, the total number of items in the result set is obtained from the SOAP response and assigned to a variable; this number is then divided by ten and rounded up to obtain the total number of pages to be displayed. Then, previous and next page links are added to the bottom of the page − each link calls the same script again and passes it a new page number via the GET method. This page number is then incorporated into the call to BrowseNodeSearchRequest(), and a new data set is obtained and displayed.

Here's what it looks like:



Brain Storm / Richard Dooling
List Price: $14.00 / Amazon.com Price: $11.20
Read more about this title on Amazon.com

Agatha Raisin and the Witch of Wyckhadden / M. C. Beaton
List Price: $21.95 / Amazon.com Price: $21.95
Read more about this title on Amazon.com

Where There's Smoke / Mel McKinney
List Price: $22.95 / Amazon.com Price: $22.95
Read more about this title on Amazon.com

A Ghost of a Chance: A Sheriff Dan Rhodes Mystery (Crider, Bill, Sheriff Dan Rhodes Books.) / Bill Crider
List Price: $23.95 / Amazon.com Price: $23.95
Read more about this title on Amazon.com

Previous page Page 2 of 5566 Next page
Disclaimer: All product data on this page belongs to Amazon.com. No guarantees are made as to accuracy of prices and information. YMMV!

One caveat, though: AWS 2.0 contains a bug that sometimes causes it to display an incorrect number of total results. Hopefully, this will be fixed in an upcoming release − until then, be warned.

Weapon Of Choice

In addition to the BrowseNodeSearchRequest() call, which is kinda like a shotgun, AWS also allows you to laser in on a specific item via the ASINSearchRequest() method, which accepts an ASIN − Amazon's unique code for each product − and returns information on the corresponding item. Consider the following example, which demonstrates:

```
<?php
// include class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient->getProxy();

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'asin' => sprintf("%010d", $_GET['asin']),
'tag' => 'melonfire−20',
'type' => 'heavy',
'devtag' => 'YOUR−TOKEN−HERE'
);

// invoke the method
$result = $proxy->ASINSearchRequest($params);

if($result['faultstring'])
{
echo $result['faultstring'];
}
else
{
$items = $result['Details'];

// print the result
print_r($result);
}
?>
```

Note the difference in the arguments passed to the method call − instead of a "browse_node" argument, this method used the "asin" argument, which specifies the ASIN to search for. This ASIN must be provided to the script above via the URL GET method, like this:

```
http://your.server/asin.script.php?asin=0735712271
```

Here's what the output of the script above looks like:

```
Array
(
[Details] => Array
(
[0] => Array
(
[Url] =>
http://www.amazon.com/exec/obidos/redirect?tag=melonfire-20%26creative=Y
OUR-
TOKEN-HERE%26camp=2025%26link_code=sp1%26path=ASIN/0735712271
[Asin] => 735712271
[ProductName] => XML and PHP
[Catalog] => Book
[Authors] => Array
(
[0] => Vikram Vaswani
)

[ReleaseDate] => 15 July, 2002
[Manufacturer] => New Riders Publishing
[ImageUrlSmall] =>
http://images.amazon.com/images/P/0735712271.01.THUMBZZZ.jpg
[ImageUrlMedium] =>
http://images.amazon.com/images/P/0735712271.01.MZZZZZZZ.jpg
[ImageUrlLarge] =>
http://images.amazon.com/images/P/0735712271.01.LZZZZZZZ.jpg
[ListPrice] => $39.99
[OurPrice] => $27.99
[UsedPrice] => $21.5
[ThirdPartyNewPrice] => $25.14
[SalesRank] => 29,310
[BrowseList] => Array
(
[0] => Array
(
[BrowseName] => XML (Document markup
language)
)

[1] => Array
(
[BrowseName] => PHP (Computer
program language
)

[2] => Array
(
[BrowseName] => Computer Programming
Languages
)

[3] => Array
(
[BrowseName] => Computer Networks
)

[4] => Array
(
[BrowseName] => Computer Bks -
Languages / Programming
)

[5] => Array
(
[BrowseName] => Computers
)

[6] => Array
(
[BrowseName] => Programming
Languages - XML
)

[7] => Array
(
[BrowseName] => Programming -
General
)

[8] => Array
(
[BrowseName] => Programming
Languages - General
)
```

```
[9] => Array
(
[BrowseName] => Programming
Languages − HTML
)

[10] => Array
(
[BrowseName] => Internet − General
)

)

[Media] => Paperback
[NumMedia] => 1
[Isbn] => 735712271
[Availability] => Usually ships within 24 hours
[Reviews] => Array
(
[AvgCustomerRating] => 3.8
[CustomerReviews] => Array
(
[0] => Array
(
[Rating] => 5
[Summary] => Lots Of Good
Content, Examples
[Comment] => i have bought
both the wrox book and this one and much prefer this one.
while the wrox
book is good, i find this one to be much easier to understand,
and to
use as a base for my own projects. i am building an XML−based
transaction server, and the chapters on DOM, WDDX and SOAP
were very
useful, as i was able to use some of the code from the book in
my
project without any difficulty. also i appreciated the
chapters on using
open−source alternatives to the built−in functions (this is
again not
available in wrox, which also tended to be infuriatingly vague
at
certain points).
```

if you are a serious developer, i would recommend
buying both books − i refer to both the wrox book and this one since
neither one is exhaustive − but i learnt more from this one, as it is
written in a clearer manner.

```
)

[1] => Array
(
[Rating] => 2
[Summary] => A questionable
book...
[Comment] => After all the
flaky reviews this book has received, I was unsure if I was reading
individual marketing campaigns sponsored by the various publishers or
actual reviews. It seems that people cannot simply agree that this book
is good or is bad as there is just nothing in between. Even in all the
review cases, many people didn't find the reviews helpful, both positive
and negative. It all seems complex from the consumer's perspective when
deciding to buy this book.

So given all these statements, I thought
I'd present a true review − one from an actual reader rather than from
someone else. I think it's pretty safe to assume that this book is good
for some people and bad for others. The problem is that the reviews
already here have so much fluff that they didn't even begin to describe
themselves, thus they could be ambitious or lazy, smart or dim, and
hobbyist or entrepreneurs. There is simply no way of telling.

Personally, I think many of these concepts can be learned in PHP in
about 2−3 days of trying the APIs out if you already know a great deal
of XML. So if I'm going to buy a book on PHP and XML, I expect that it
will provided added value information as well as design decisions,
business concerns and best practices. Examples are not what I care about
as much as the rich and deep information because there are many examples
already on the web − no point acquiring the book just for those alone.
That makes me question the reviewers who say the examples are clear and
concise − the examples on the web already do that. Books are supposed to
provide added value to these APIs and examples to make the topic
complete and valuable to the reader. The book should also scale well to
both beginning audiences (this book does very well) to expert audiences
that want to drill through the basic information like APIs and examples
and learn more advanced techniques, best practices, etc. This book
doesn't deliver on these areas very well unfortunately.

So, for a
person like me: This book receives 2 stars. I didn't learn all that much
from it and I was disappointed to say the least. He's a good writer,
funny at times, and knows what he is doing, but he also catered to a
specific audience and it shows. Is that the goal? Probably. But I think
the expert people shouldn't have expected too much (as I did) − that's
the truth.

Although I personally give this book 2 stars, I believe
```

```
that many beginner PHP programmers who have a little idea to what XML is
will benefit from it. If you've already read some XML material on the
net and even read a book or two, this book won't exactly help you out
too much. However, this segment is rather small I would believe. I'm
still looking for a book that I can give to my employees for reference
as well as added value information. When I find it, I'll put a review
there as well so you can compare.

So there you have it − an honest
review. I hope it helps people out in their purchasing decisions.
)

[2] => Array
(
[Rating] => 4
[Summary] => One of the best
XML and PHP titles
[Comment] => Most XML books
suffer from painful verbosity. Useful information on a relatively simple
subject tends to be hidden in drifts of useless cruft.

This book
focuses on the core information needed to become competent using XML and
PHP together. While it is not the most comprehensive reference on XML
available, it is the first resource that I check.

(Disclosure: I
worked on this book project as a technical reviewer − take what I say
with your own grain of salt. :)
)

)

)

[SimilarProducts] => Array
(
[0] => 1861007213
[1] => 1861006918
[2] => 073570970X
[3] => 672317842
[4] => 1565926102
)

)

)

)
```

Wondering how I got so much extra data this time? That's because I told AWS I wanted the "heavy" form of the data, not the "lite" one I've been using thus far.

In case you're curious, yes, I've used the sprintf() function call to reformat the ASIN to a ten−character zero−padded string. This is because AWS will return an error if the ASIN passed to it in a method call is less than ten characters in length.

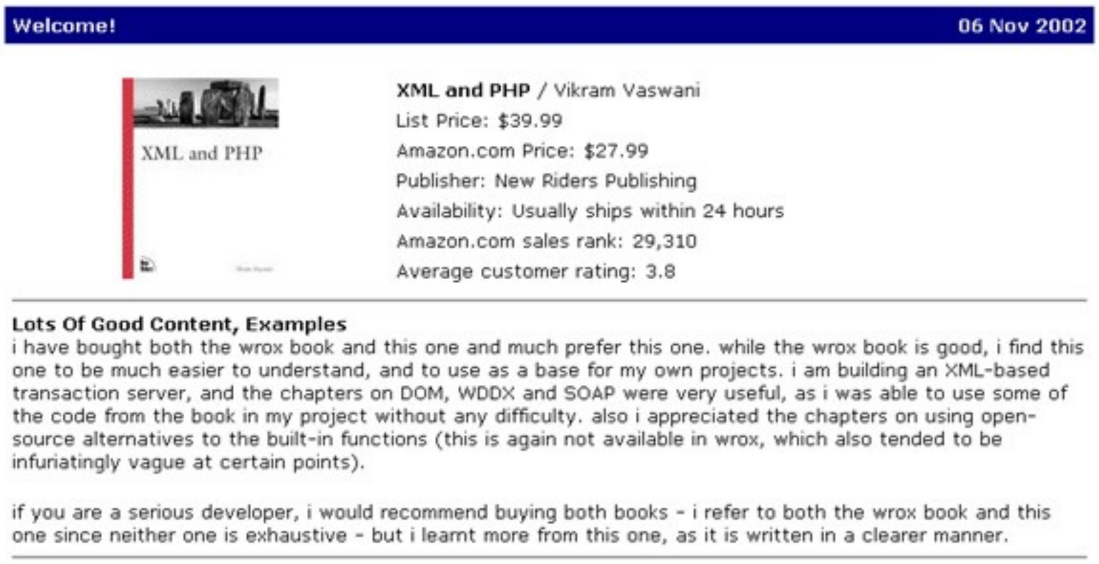
Hooking Up

The ASINSearchRequest() method, combined with the AWS "heavy" data format, makes it easy to build detailed product information pages that are similar to the originals on Amazon.com. Consider the following revision to the example on the previous page, which accepts an ASIN on the URL and returns a neatly−formatted product information page:

```
<?php

// include class
include("nusoap.php");

// create a instance of the SOAP client object
$soapclient = new
soapclient("http://soap.amazon.com/schemas2/AmazonWebServices.wsdl",
true);

// uncomment the next line to see debug messages
// $soapclient->debug_flag = 1;

// create a proxy so that WSDL methods can be accessed
directly
$proxy = $soapclient->getProxy();

// set up an array containing input parameters to be
// passed to the remote procedure
$params = array(
'asin' => sprintf("%010d", $_GET['asin']),
'tag' => 'melonfire−20',
'type' => 'heavy',
'devtag' => 'YOUR−TOKEN−HERE'
);

// invoke the method
$result = $proxy->ASINSearchRequest($params);
$items = $result['Details'];

// display the result
?>

<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">
```

```
<p>&nbsp;<p>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td bgcolor="Navy"><font color="white" size="−1"><b>Welcome to
The Mystery Bookstore!</b></font></td>
<td bgcolor="Navy" align="right"><font color="white"
size="−1"><b><? echo date("d M Y",
mktime());?></b></font></td> </tr>
</table>

<p>

<table width="100%" border="0" cellspacing="5"
cellpadding="0"> <tr> <td
align="center" valign="top" rowspan="7"><a href="<? echo
$items[0]['Url']; ?>"><img border="0" src=<? echo
$items[0]['ImageUrlMedium']; ?>></a></td> <td><font
size="−1"><b><? echo
$items[0]['ProductName']; ?></b> / <? echo implode(", ",
$items[0]['Authors']); ?></font></td> </tr> <tr> <td
align="left"
valign="top"><font size="−1">List Price: <? echo
$items[0]['ListPrice'];
?></font></td> </tr> <tr> <td align="left" valign="top"><font
size="−1">Amazon.com Price: <? echo $items[0]['OurPrice'];
?></font></td> </tr> <tr> <td align="left" valign="top"><font
size="−1">Publisher: <? echo $items[0]['Manufacturer'];
?></font></td>
</tr> <tr> <td align="left" valign="top"><font
size="−1">Availability:
<? echo $items[0]['Availability']; ?></font></td> </tr> <tr>
<td
align="left" valign="top"><font size="−1">Amazon.com sales
rank: <? echo
$items[0]['SalesRank']; ?></font></td> </tr> <tr> <td
align="left"
valign="top"><font size="−1">Average customer rating: <? echo
$items[0]['Reviews']['AvgCustomerRating']; ?></font></td>
</tr> <tr> <td
colspan="2"> <font size="−1"> <hr> <?
foreach($items[0]['Reviews']['CustomerReviews'] as $r)
{
?>
<b><? echo $r['Summary']; ?></b>
<br>
<? echo $r['Comment']; ?>
<hr>
<?
}
?>
</font>
</td>
</tr>
</table>

<font size="−1">
Disclaimer: All product data on this page belongs to
Amazon.com. No
guarantees are made as to accuracy of prices and information.
YMMV!
</font>

</body>
</html>
```

Here's what it looks like:


Welcome! 06 Nov 2002

XML and PHP / Vikram Vaswani
List Price: $39.99
Amazon.com Price: $27.99
Publisher: New Riders Publishing
Availability: Usually ships within 24 hours
Amazon.com sales rank: 29,310
Average customer rating: 3.8

Lots Of Good Content, Examples
i have bought both the wrox book and this one and much prefer this one. while the wrox book is good, i find this one to be much easier to understand, and to use as a base for my own projects. i am building an XML-based transaction server, and the chapters on DOM, WDDX and SOAP were very useful, as i was able to use some of the code from the book in my project without any difficulty. also i appreciated the chapters on using open-source alternatives to the built-in functions (this is again not available in wrox, which also tended to be infuriatingly vague at certain points).

if you are a serious developer, i would recommend buying both books - i refer to both the wrox book and this one since neither one is exhaustive - but i learnt more from this one, as it is written in a clearer manner.

Pretty cool, huh?

Now here's something for you to think about. Remember how, a few pages back, I built a product catalog with the BrowseNodeSearchRequest() method and linked each item in that catalog to the actual product page on Amazon.com? Well, with the ASINSearchRequest() method, you no longer need to link to Amazon.com for detailed product information − you can generate it yourself! Simply alter the links in the product catalog to point to the PHP script above, send the script the ASIN via the GET method, and you can provide your visitors with detailed product information on your own site.

I'll leave this last to you as an exercise. Give it a shot, and come back next week for the second part of this article, when I'll be showing you how to add search capabilities to your rapidly−evolving online store. See you then!

Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.24, PHP 4.2.3, NuSOAP 6.3 and AWS 2.0. Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. All product data in this article belongs to Amazon.com. YMMV!