By icarus

Table of Contents

The Road Ahead

In the first part of this article, I introduced you to Amazon Web Services (AWS), the XML−based API that allows regular Joes to supercharge their online stores with Amazon's product catalog and databases. I showed you the basics of AWS, demonstrating the API calls needed to build a product listing and obtain detailed product information on any item in Amazon's catalog.

In this concluding part, I'm going to spend some time discussing the search features built into AWS, showing you the various types of search options available and illustrating, with examples, how they can be used to improve the user experience at your store. I'll also show you how to link your store up to Amazon's transaction system, by adding support for both shopping carts and wish lists to your site. All that and more, inside!

Rolling The Stones

The first − and the simplest − type of search available in AWS is the keyword search, exposed via the KeywordSearchRequest() call. Here's a simple example of how it works:

```
<?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);

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

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

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

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

This is functionally similar to the scripts you've seen thus far; the primary difference lies in the call to a new function − KeywordSearchRequest() − and in the different arguments passed to it. The "keyword" argument passed to the function contains the words to be used as search keys in the specified Amazon catalog(s); note that this search term must be URL-encoded before being passed to the function.

Here's sample output from the script above:

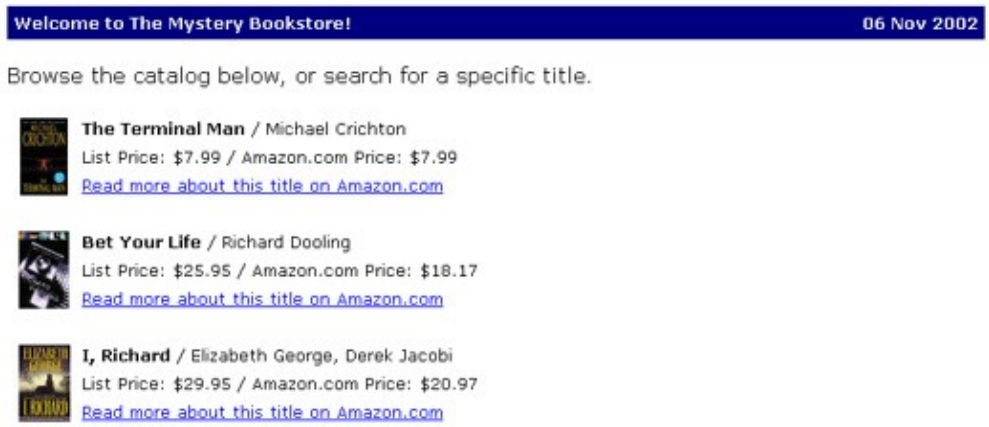
```
Array
```

```
(
[TotalResults] => 163
[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/1556523734
[Asin] => 1556523734
[ProductName] => Nankering With the Rolling Stones
[Catalog] => Book
[Authors] => Array
(
[0] => James Phelge
)

[ReleaseDate] => April, 2000
[Manufacturer] => A Cappella Books
[ImageUrlSmall] =>
http://images.amazon.com/images/P/1556523734.01.THUMBZZZ.jpg
[ImageUrlMedium] =>
http://images.amazon.com/images/P/1556523734.01.MZZZZZZZ.jpg
[ImageUrlLarge] =>
http://images.amazon.com/images/P/1556523734.01.LZZZZZZZ.jpg
[ListPrice] => $16.95
[OurPrice] => $11.87
[UsedPrice] => $10.95
)

... snip ...

)

)
```

Next, let's see how this can be integrated into the sample store I've been building.

Searching For Words

You might remember this page from the first part of this article:



You might also remember that I never got around to adding a search box. Let's do that now:

```
<?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);

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

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

// if keyword available, use it
if ($_POST['keyword'] && $_POST['keyword'] != "") { $keyword =
$_POST['keyword']; } if ($_GET['keyword'] && $_GET['keyword']
!= "") {
$keyword = $_GET['keyword']; }

// if a keyword is present, do a keyword search
if ($keyword)
{
```

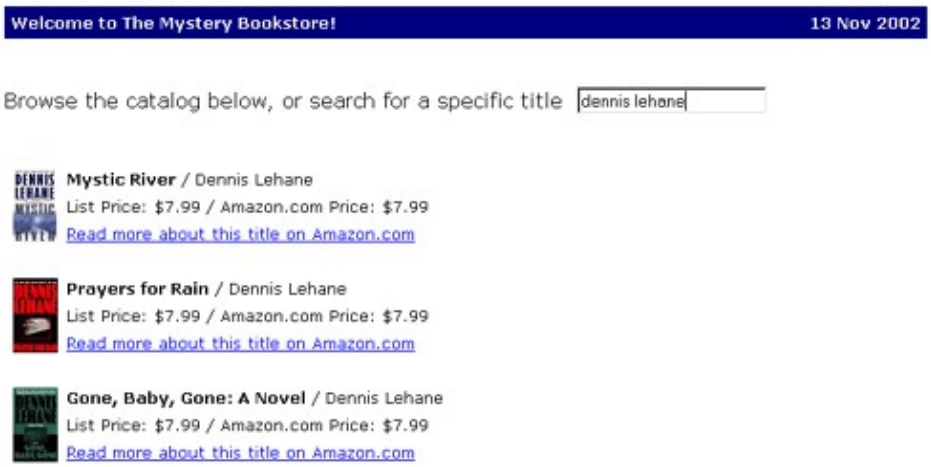
```
$params = array(
'keyword' => htmlentities($keyword),
'page' => $page,
'mode' => 'books',
'tag' => 'melonfire-20',
'sort' => '+pmrank',
'type' => 'lite',
'devtag' => 'YOUR-TOKEN-HERE'
);

// invoke the method
$result = $proxy->KeywordSearchRequest($params);
}
else
{
// else perform a node browse
$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> <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 border="0" cellspacing="0" cellpadding="0">
<tr>
<td>Browse the catalog below, or search for a specific title
 </td>
<td><br><form method="post" action="<? echo
$_SERVER['PHP_SELF'];
?>"><input type="text" name="keyword"></form></td> </tr>
</table>

<p>

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

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> </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;
?>&keyword=<? echo $keyword; ?>">Previous page</a> <? } ?>
  </td>
<td align="center">Page <? echo $page; ?> of <? echo
$pageCount; ?></td>
<td align="right">   <? if ($page < $pageCount) { ?> <a
href="<?
echo $_SERVER['PHP_SELF']; ?>?page=<? echo $page+1;
?>&keyword=<? echo
$keyword; ?>">Next page</a> <? } ?> </td> </tr> </table>

</body>
</html>
```

As you can see, this is pretty simple − I've added a small input box for the user to enter a search term, and modified the form processor to check for a keyword and run a KeywordSearchRequest() if it's present, or a BrowseNodeSearchRequest() for the default node if it isn't. The next and previous page links also now pass an additional "keyword" parameter back and forth, so that the user can page through either the product catalog returned by BrowseNodeSearchRequest() or the result set returned by KeywordSearchRequest().

Here are a couple of screenshots demonstrating what the final result looks like:



Drilling Deeper

AWS also allows you to look for items in specific categories, supporting searches by author, actor, artist and manufacturer. Here's a list of the relevant methods:

AuthorSearchRequest() − search by author;

ArtistSearchRequest() − search by artist or musician;

ActorSearchRequest() − search by actor or actress;

DirectorSearchRequest() − search by director;

ManufacturerSearchRequest() − search by product manufacturer;

With methods like these, it's easy to build a more powerful search engine for your store. Take a look:

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

<body bgcolor="white">

<p> <p>

<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td bgcolor="Navy"><font color="white"
size="−1"><b>Search</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>

<?
if (!$_POST['q'])
{
?>

Select a category and enter a search term:
<form method="post" action="<? echo $_SERVER['PHP_SELF']; ?>">
<select
name="type">
<option value="author">Author</option>
```

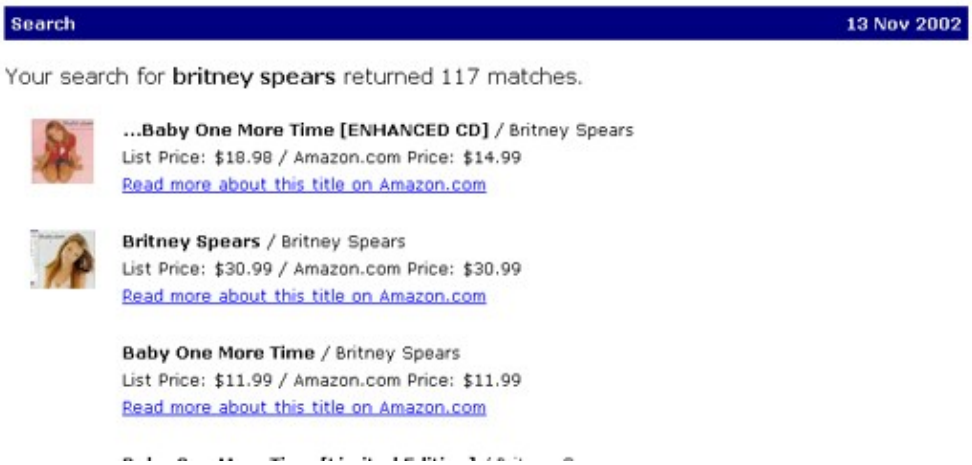
```
<option value="artist">Artist</option>
<option value="actor">Actor</option>
<option value="director">Director</option>
<option value="manufacturer">Manufacturer</option>
</select>
<input type="text" name="q">
</form>

<?
}
else
{
$type = $_POST['type'];

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

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

// 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
// use a switch loop to define the search parameters for each
type

switch ($type)
{
case "author":
$mode = "books";
$func = "AuthorSearchRequest";
break;

case "artist":
$mode = "music";
$func = "ArtistSearchRequest";
break;

case "actor":
$mode = "vhs";
$func = "ActorSearchRequest";
break;

case "director":
```

```
$mode = "vhs";
$func = "DirectorSearchRequest";
break;

case "manufacturer":
$mode = "electronics";
$func = "ManufacturerSearchRequest";
break;
}

$params = array(
$type => htmlentities($_POST['q']),
'page' => 1,
'mode' => $mode,
'tag' => 'melonfire-20',
'type' => 'lite',
'devtag' => 'YOUR-TOKEN-HERE'
);

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

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

// format and display the results
?>

Your search for <b><? echo $_POST['q']; ?></b> returned <?
echo $total;
?> matches.

<p>

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

// parse the $items[] array and extract the necessary
information
foreach ($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> /
<?
```

```
if (is_array($i['Authors']))
{
echo implode(", ", $i['Authors']);
}
else if (is_array($i['Artists']))
{
echo implode(", ", $i['Artists']);
}
else if ($i['Manufacturer'])
{
echo $i['Manufacturer'];
}?>
</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> </td> </tr> <?
} ?> </table>

<?
}
?>

</body>
</html>
```

In this case, I've simply added a new item to my search form − a drop−down list containing a list of the categories to search in. Each of these categories corresponds to one of the AWS methods listed above, and my PHP form processor uses a "switch" statement to call the appropriate method and pass it the search keywords. The result is then processed in the usual way, and formatted for display to the user.

Here are a couple of screenshots demonstrating how this works:


Search
13 Nov 2002
Your search for britney spears returned 117 matches.
...Baby One More Time [ENHANCED CD] / Britney Spears
List Price: $18.98 / Amazon.com Price: $14.99
Read more about this title on Amazon.com
Britney Spears / Britney Spears
List Price: $30.99 / Amazon.com Price: $30.99
Read more about this title on Amazon.com
Baby One More Time / Britney Spears
List Price: $11.99 / Amazon.com Price: $11.99
Read more about this title on Amazon.com

Riding The Popularity Metrics

One of the things I've always liked about Amazon.com is its recommendations system − each time a customer makes a purchase, the Amazon.com system registers it and, keeping the customer's past purchases in context, evolves a database of customer likes and dislikes. This database is then used to provide other customers with a list of similar items ("...customers who purchased this item also bought..."), thereby serving the role of a very experienced sales person. It's a great system, works like a charm...and is now available to you via the AWS SimilaritySearchRequest() call.

In order to demonstrate how this works, consider the following script:

```
<?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']),
'mode' => 'books',
'tag' => 'melonfire-20',
'type' => 'heavy',
'devtag' => 'YOUR-TOKEN-HERE'
);

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

$simResult = $proxy->SimilaritySearchRequest($params);
$simItems = $simResult['Details'];


// display the result
```

```
?>

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

<body bgcolor="white">

<p> <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></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 $r['Comment']; ?>
<hr>
<?
}
?>
</font>
</td>
</tr>
<tr>
<td colspan="2">
<font size="-1">
Customers who shopped for this item also bought:
<ul>
<?
foreach($simItems as $s)
{
echo "<li><a href=\"" . $s['Url'] . "\">" . $s['ProductName']
.
"</a>"; } ?> </ul></font> </td> </tr> </table>

</body>
</html>
```

The key to this script is the SimilaritySearchRequest() method call − this method accepts an ASIN (Amazon.com's unique code for each product) and returns a list of similar items, based on Amazon's customer purchase statistics, together with brief descriptive information. This information can then be formatted and displayed as a neat little addition to your product catalog − exactly what I've done above

With the addition of the call to SimilaritySearchRequest(), the page now also includes helpful data on other customer's past purchases in the same context, together with links to the relevant product pages (notice how I'm using the returned ASIN's to re−invoke the same PHP script again, but with a different ASIN).

Bagging It

Finally, after providing you with the API to build a product catalog, search for products, and obtain detailed product descriptions, AWS rounds things out by allowing you an entry point into Amazon's transaction system, providing constructs to add products to an Amazon.com shopping cart or wish list.

For the first time in a while, accomplishing this doesn't involve using any special SOAP commands − all you need is a plain ol' HTML form, containing certain mandatory fields.

```
<form method="POST"
action="http://www.amazon.com/o/dt/assoc/handle-buy-box=<ASIN>">
<input type="hidden" name="asin. <ASIN>" value="1">
<input type="hidden" name="tag-value"
value="<YOUR-ASSOCIATES-ID>">
<input type="hidden" name="tag_value"
value="<YOUR-ASSOCIATES-ID>">
<input type="hidden" name="dev-tag-value"
value="<YOUR-TOKEN>"> <input
type="submit" name="submit.add-to-cart" value="Buy From
Amazon.com"> or
<input type="submit" name="submit.add-to-registry.wishlist"
value="Add
to Amazon.com Wish List">
</form>
```

So, if you wanted to place a "Buy!" link on each product page for the book "XML and PHP" − Amazon.com ASIN 0735712271 − your form would look like this:

```
<form method="POST"
action="http://www.amazon.com/o/dt/assoc/handle-buy-box=0735712271">
<input type="hidden" name="asin.0735712271" value="1">
<input type="hidden" name="tag-value" value="melonfire-20">
<input
type="hidden" name="tag_value" value="melonfire-20"> <input
type="hidden" name="dev-tag-value" value="YOUR-TOKEN-HERE">
<input
type="submit" name="submit.add-to-cart" value="Buy From
Amazon.com"> or
<input type="submit" name="submit.add-to-registry.wishlist"
value="Add
to Amazon.com Wish List">
</form>
```

Needless to say, it's pretty easy to incorporate this into the product information page designed in the previous example − here's the script:

```
<?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);

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

// format ASIN to ten-character format
$asin = sprintf("%010d", $_GET['asin']);

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

// get item details
$result = $proxy->ASINSearchRequest($params);
$items = $result['Details'];

// get list of similar items
$simResult = $proxy->SimilaritySearchRequest($params);
$simItems = $simResult['Details'];


// display the result
?>

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

<body bgcolor="white">
```

```
<p> <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="8"><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></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
align="left" valign="top"><font size="-1"> <form method="POST"
action="http://www.amazon.com/o/dt/assoc/handle-buy-box=<?
echo $asin;
?>"> <input type="hidden" name="asin.<? echo $asin; ?>"
value="1">
<input type="hidden" name="tag-value" value="melonfire-20">
<input
type="hidden" name="tag_value" value="melonfire-20"> <input
type="hidden" name="dev-tag-value" value="YOUR-TOKEN-HERE">
```

```
<input
type="submit" name="submit.add-to-cart" value="Buy From
Amazon.com"> or
<input type="submit" name="submit.add-to-registry.wishlist"
value="Add
to Amazon.com Wish List">
</form></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 $r['Comment']; ?>
<hr>
<?
}
?>
</font>
</td>
</tr>
<tr>
<td colspan="2">
<font size="-1">
Customers who shopped for this item also bought:
<ul>
<?
foreach($simItems as $s)
{
echo "<li><a href=\"" . $_SERVER['PHP_SELF'] . "?asin=" .
$s['Asin'] . "\">" . $s['ProductName'] . "</a>"; }

?>
</ul></font>
</td>
</tr>
</table>

</body>
</html>
```

And here's an example of the page generated:

Clicking either of the form buttons will redirect the user to Amazon.com, where the selected item gets added to a shopping cart/wish list. Consummating the transaction is currently outside the scope of AWS, and is handled by the main Amazon.com site; if you're an Amazon.com Associate, your account will get credited with appropriate commission, since the URL passed to Amazon.com includes your Associate ID. It all fits together pretty neatly.

Linking Out

In addition to the methods described above, AWS includes a few more − here's a list of the ones I've missed:

SellerSearchRequest() − retrieve a list of products sold by third−party sellers

ListManiaSearchRequest() − retrieve a list

WishlistSearchRequest() − retrieve a wishlist

SellerProfileSearchRequest() − retrieve a third−party seller profile

Integrating these into your online store is not very difficult − the procedure is technically similar to what is described above, with appropriate changes to the function calls and responses. For more information, take a look at the following links:

The AWS home page, at http://www.amazon.com/webservices/

The AWS discussion boards, at http://forums.prospero.com/am−assocdevxml

The AWS "lite" DTD, at http://xml.amazon.com/schemas2/dev−lite.dtd

The AWS "heavy" DTD, at http://xml.amazon.com/schemas2/dev−heavy.dtd

The SOAP WSDL, at http://soap.amazon.com/schemas2/AmazonWebServices.wsdl

And that's about it for the moment. In this concluding article, I introduced you to the AWS methods that allow you to add search functionality to your Amazon−backed online store, demonstrating how they could be used to locate items by keyword, author, artist, actor or manufacturer. I also showed you how to refine searches using similarity tests, and illustrated how you can provide shoppers with the ability to buy items using Amazon.com Payments, or add them to a wishlist on the main Amazon.com site.

I hope you enjoyed this article, and that you found it helpful in your online e−commerce experience. Let me know what you thought, or if you'd like to read more about AWS...and until I see you next, have fun!

Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.24, PHP 4.2.3, NuSOAP 6 (rev 1.11) 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!