


[Community Recognition](#) | [Guidelines](#) | [Help](#) | |

[Resource Center](#) | [Forums](#) | [Solutions Catalog](#) | [Co-marketing](#) | [Newsletter](#) | [Blog](#) | [Support](#)


Advanced

[Amazon Web Services](#) » [Developer Connection](#) » ... » [Amazon SimpleDB \(Beta\)](#) » [Articles and Tutorials](#) » [Creating a WordPress Plugin Using Amazon SimpleDB](#)

Creating a WordPress Plugin Using Amazon SimpleDB

[« Back to Search Results](#)
[Printer Friendly](#) | [Save to del.icio.us](#)

Thomas Myer guides you through the steps of building a WordPress plugin that works with Amazon SimpleDB to tag blog posts.

[Discussion](#)
[Reviews](#)

AWS Products Used:	Amazon SimpleDB
Language(s):	PHP
Date Published:	2008-04-14

By [Thomas Meyer](#)

Unless you've been on a 5-year vacation to the moon or been completely unplugged from the most recent discourse on the Internet, you know that tagging is a buzzword du jour. Just in case, let's define the word: Tagging is the activity of adding informational keywords (aka *tags*) to online content to make it easier to find or categorize.

What's the point? Although much of the content on the web is text-based and therefore fairly easy for search engines such as Google to index, a fast-growing portion of web content isn't textual. Movies, pictures, audio files, games, and other content are more difficult to index; tagging this content makes the job easier. Tagging can add value and insight for users trying to find content related to a particular keyword.

Critics may claim that tags are inherently personal and therefore flawed. For example, try running a Flickr search for a particular keyword and see how many irrelevant pictures you wind up trawling through. But tags, inconsistent as they may be, can provide insight and structure to a group of documents that would otherwise seem unrelated.

Starting with WordPress version 2.3, bloggers on the platform can easily and natively add tags to any post. In the past, the tagging was accomplished solely by using categories or third-party plugins. I'm going to teach you the basics of WordPress plugin creation. Then you'll create a specific plugin that works with Amazon Web Services (AWS) SimpleDB, which gives you access to a simple and powerful database that you can use to aggregate your blog tags. You'll build a related-content list to display tagged content. Finally, you'll build a simple Options menu to help manage your plugin.

Why Amazon SimpleDB? What's the point of storing your blog's tags at Amazon? Many of you run more than one blog—I meet many people who are building mini-media conglomerates with three, four, or even a dozen blogs. Those are a lot of tags to manage; keeping them in one place simplifies the chore. Even if you have only one blog, you can use Amazon SimpleDB as a backup storage system for key blog data.

What is Amazon SimpleDB?

According to the AWS site, "Amazon SimpleDB is a web service for running queries on structured data in real time." Basically, you can store and retrieve data records that consist of name-value pairs. These data records reside in domains that you stipulate. The end result is similar to the data stored in spreadsheets—something that may take a little getting used to if you're a relational-database person.

For example, you might create a domain called "to do list," then use it to store items that relate to tasks you need to complete. You might create a task named "review article," paired with a URL that points to the article. Each task corresponds to a cell in a spreadsheet. However, unlike in a spreadsheet, you can add as many as 256 name-value pairs to any individual cell, with each value containing as much as 1024 bytes.

How to Sign Up

Before you can use any AWS product, you need to sign up. The process takes about 5–10 minutes and requires you to fill out a simple form and register a credit card to pay for services.

Simply go to <http://aws.amazon.com> to sign up. Remember, to use any of this code, you need an Access Key ID and a Secret Access Key from Amazon. You'll get both via email as soon as you complete your registration.

Once you have an AWS account, you must subscribe to the Amazon SimpleDB public beta. (The service eventually will be open to everyone.)

Download the Amazon SimpleDB Tools

Welcome, Guest

[Login](#)
 [Guest Settings](#)

After you've set up your Amazon SimpleDB account, run a search for the Amazon SimpleDB tools and add them to your web server. Make sure that you place them in a directory that is visible to your path (you'll be using a require statement to pull them in), or add the folder you use to your path. The download contains a collection of PHP SimpleDB classes and examples that will help you during this project and future projects involving Amazon SimpleDB.

Creating a WordPress Plugin

When WordPress first debuted, expert users were soon modifying it by digging around inside the core code. This idea was a bad one, primarily because it caused a great deal of headache whenever users upgraded to a newer version of WordPress. All that hard work was lost.

Currently, if you want to change the way that WordPress behaves (or you want to add new behaviors to WordPress), you write a plugin and store that plugin in the `wp-content/plugins` folder. That way, your work stays separate from the core code, leaving less opportunity for the curious and well-meaning to disrupt core blogging functions.

So, what is a plugin? A WordPress plugin consists of one or more PHP functions that extend WordPress in some way. Each plugin must contain header information so that it can be identified as a plugin and enabled through the WordPress admin panel. You must be careful not to allow any spaces before or after the PHP open and close tags; such spaces will cause fatal errors. Other than that, there aren't too many rules—you could create a plugin that converts every word in a blog post to "foo." We won't be that capricious.

The first step in creating a WordPress plugin is to create a file with a unique name. For this exercise, you'll use the name `simpledb-tag.php`. Use your favorite text editor to create a file with this name, then put the following code into the file:

```

Plugin Name: Amazon SimpleDB Tagging
Engine

Version: 0.1

Plugin URI:

Description: Creates a tagging
engine on Amazon SimpleDB

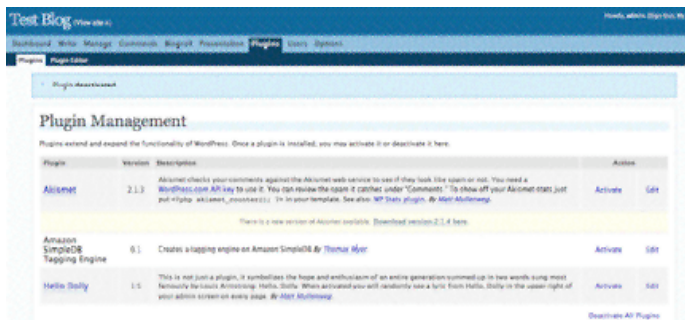
Author: Thomas Myer

Author URI:
http://www.tripledogs.com/

*/

```

Save the file, then upload it to your blog's `wp-content/plugins` folder. Log into your WordPress admin panel, go to the Plugins tab, and activate the file. Notice that in the following screenshot, the Plugin Management screen displays the information from the plugin's comment header.



Of course, activating your plugin at this point is fairly meaningless: It doesn't do anything yet. We're about to change that, but we need an active plugin with which to work.

Understanding WordPress 2.3 Tagging

When a WordPress 2.3 author decides to add tags to a post, the interface is fairly simple. You write the post, add tags in a separate field, and then publish the post. End of story, right?

Not by a long shot. It doesn't take much digging in the database backend to realize that a lot happens once a post is tagged. Take a deep breath and follow along. You need to understand this to follow the rest of the process.

After you write and tag a post, the following events happen behind the scenes:

1. Blog posts are stored in a table called `wp_posts` (unless you've changed the prefix to something else during install; this exercise assumes that you've kept the defaults).
2. Blog post tags are stored in a table named `wp_terms`.
3. The relationship between `wp_posts` and `wp_terms` is kept in a table called `wp_term_relationships`.

(where `object_id` maps to the `wp_posts` ID field and `term_taxonomy_id` maps to the `term_id` in `wp_terms`).

4. But that's not all! Another table, called `wp_term_taxonomy`, maps to `wp_terms`. This table provides an all-important taxonomy field that indicates the kind of content to which the tag is applied (for this example, you'll care only about post_tag content).

You must perform a four-table join to have any hope of figuring out which tags go with which posts. Luckily, you're going to make your main function do its work whenever a new post is added to a site (or refreshed, once the post is published), so you'll deal with only one set of parameters at a time.

Your SQL query will end up looking something like the following:

```
select a.ID,a.post_title,c.name as tag
from wp_posts a,
wp_term_relationships b, wp_terms c, wp_term_taxonomy
d
where a.ID = b.object_id
and b.term_taxonomy_id = c.term_id
and c.term_id = d.term_id
and d.taxonomy = 'post_tag'
and a.post_status='publish'
and a.ID = X
```

...where `X` is the ID of the post with which you're dealing. (If you want to take a break at this point, go ahead. I'll be here when you get back.)

Building Your First Function

Let's take what you've learned about the taxonomical structure of WordPress 2.3 tagging and create a plugin function around it.

The first thing you're going to do with your plugin is create some variables that you can then pass in as global functions. You'll need to use this information often as you work, so you might as well lay the groundwork now.

```
//authentication
$AWS_ACCESS_KEY_ID      = "change-me";
$AWS_SECRET_ACCESS_KEY  = "change-me-too";
$domain = 'my_blog_tags';
```

You'll use the Access Key ID and Secret Access Key to log in to Amazon SimpleDB. The domain will keep your data focused on one area.

The first thing to realize is that the function on which you're working will be run every time someone publishes a post on the blog. So, you need to instantiate the function and pass in a post ID:

```
function sdb_publish_tags($post_id){
}
```

Next, you need to pass in the special `$wpdb` object, which contains all the database connectivity you need to do your work. You should also set some variables and arrays that you'll need in a bit, and you'll use `get_option()` to grab the site URL of your blog (this value is set during install). You're potentially dealing with numerous blogs, so you want to be able to identify where a post came from.

```
function sdb_publish_tags($post_id){
    global $wpdb;
    $stat = 'publish';
    $tax = 'post_tag';
    $url = get_option('siteurl');
}
```

Next, you'll insert and run your four-table join query. Instead of hard-coding `wp_` as the table prefix, you'll use the more acceptable `$wpdb->`, which allows the SQL code to keep up with any table name changes you may make across your blogs. Now is also the time to pull in the other variables you created earlier.

```
function sdb_publish_tags($post_id){
    global $wpdb;
    global $domain;
    global $AWS_ACCESS_KEY_ID;
    global $AWS_SECRET_ACCESS_KEY;
    $stat = 'publish';
```

```

    $tax = 'post_tag';
    $url = get_option('siteurl');

    $sql = "select a.ID,a.post_title,c.name as tag
            from $wpdb->posts a, $wpdb->term_relationships b, $wpdb->ter
            where a.ID = b.object_id
            and b.term_taxonomy_id = c.term_id
            and c.term_id = d.term_id
            and d.taxonomy = '$tax'
            and a.post_status='$stat'
            and a.ID = $post_id";

    $tags = $wpdb->get_results($sql);
}

```

At the end of this process, you have a result set called `$tags` to process. The easiest way is to loop through `$tags` as you would any array, and then spit out the results. Remember that you will grab a unique post title and ID but that you may have multiple keywords. Simply tuck those keywords into an array you can process later:

```

foreach ((array) $tags as $t){
    $ID = $t->ID;
    $TITLE = $t->post_title;
    $TAGS[] = $t->tag;
}

```

Adding the Amazon SimpleDB Portions

Now that you've extracted the tags from a post, it's time to post this information to Amazon SimpleDB. Each time you access Amazon SimpleDB, you will need to authenticate, using your unique access key and secret password.

You will also need to know which domain you're accessing and with which item you're working. Both the domain and the item name need to be unique to your user account. In this case, the item name you'll create will be based on your permalink, post ID, and blog's URL (which should provide a sufficiently unique item name). I always use the `/post/X` permalink structure; you can adjust as necessary to meet your needs.

You may wonder: Why not use the URL as the domain name and the post ID as the item name? That's a good idea, but if you have numerous blogs, you'll likely have competing post IDs—for example, a post ID 3 or 4 on numerous blogs—thereby destroying some data integrity. Best to use a man-made domain name to hold all the tag data from all your blogs, and use the URL in the item name to enforce uniqueness.

Right after the foreach loop that processes the tags, start laying the foundations of the integration with Amazon SimpleDB:

```

//will use this in our code below
//adjust $item_name to match your permalink structure
$item_name = $url . "/post/". $ID;

//1. login
require_once 'Amazon/SimpleDB/Client.php';
$service = new Amazon_SimpleDB_Client($AWS_ACCESS_KEY_ID, $AWS_SECRET_

```

With just a few lines of code, the Amazon SimpleDB Client authenticates against the service. The next step is to create a domain. But because you don't want to take the time and effort to create a domain each time someone publishes a blog post, it's best to determine whether the domain has already been created. (If you have this plugin working on various blogs, chances are that your domain is already in use.)

The best way to determine the existence of a domain is to use the client's built-in `listDomains()` method. After you've extracted a list of domains, you can use other helpful methods—such as `getListDomainsResult()` and `getDomainName()` to loop through the result set and find your domain.

If your domain exists, move on; if it doesn't, create it.

```

//2. make sure that our domain is created
$found_domain =false;
$response = $service->listDomains();
if ($response->isSetListDomainsResult()) {
    $listDomainsResult = $response->getListDomainsResult();
    $domainNameList = $listDomainsResult->getDomainName();
    foreach ($domainNameList as $domainName) {

```

```

        if ($domainName == $domain){
            $found_domain = true;
        }
    }
    //3. looks like we need to create the domain
    if ($found_domain == false){
        $data = array('DomainName' => $domain);
        $response = $service->createDomain($data);
    }
}

```

Finally, it's time to post the information you've extracted from the WordPress post to Amazon SimpleDB. There are two ways to do this. The first uses some helper objects, and the second involves using array notation. I feel most comfortable with array notation, so that's what I'll use in this example.

All you need to do is loop through the \$TAGS array that you built earlier, creating requests as you go and then posting that packet to Amazon SimpleDB with a try/catch. If a request fails, catch the exception and print out all the debugging information you need to fix the problem:

```

//4. Post attributes
foreach ($TAGS as $tag){
    $putAttributesRequest =
        array ("DomainName" => $domain,
            "ItemName" => $item_name,
            "Attribute" =>
                array(
                    array ("Name" => "id", "Value" => $ID),
                    array ("Name" => "title", "Value" => $TITLE),
                    array ("Name" => "tag", "Value" => $tag),
                )
        );
    try {

        $response = $service->putAttributes($putAttributesRequest);

    } catch (Amazon_SimpleDB_Exception $ex) {

        echo("Caught Exception: " . $ex->getMessage() . "\n");
        echo("Response Status Code: " . $ex->getStatusCode() . "\n");
        echo("Error Code: " . $ex->getErrorCode() . "\n");
        echo("Error Type: " . $ex->getErrorType() . "\n");
        echo("Box Usage: " . $ex->getBoxUsage() . "\n");
        echo("Request ID: " . $ex->getRequestId() . "\n");
        echo("XML: " . $ex->getXML() . "\n");
    }
} //end foreach

```

There's one final step at this stage. You must add a hook at the end of your plugin file to tell the plugin how to apply your new function.

In this case, you want the `sdb_publish_tags()` function to run every time someone publishes a post or updates a live post. Simply add this line to the end of the file (i.e., outside the function you just created):

```
add_action('publish_post', 'sdb_publish_tags');
```

Now every time a publish action happens, the `sdb_publish_tags()` function will run, adding your tags to the Amazon SimpleDB domain that you've created.

Creating a Related Content List

Now that you're saving tags inside Amazon SimpleDB, it's time to create a basic function that will query Amazon SimpleDB and construct a related content list at the end of any blog post. The idea is to take the tags listed for a blog post and then retrieve posts stored in Amazon SimpleDB and that match those tags.

You'll call this new function `sdb_related_content()`. In many ways, it will be similar to the first function, except this time you'll use the global `$post` variable to help extract the ID of the blog post in the user's browser. With that ID, you can then extract tags and do our other work.

The first part of the function is all about retrieving tags:

```

function sdb_related_content(){
    global $wpdb, $wp_query;
    global $domain;
    global $AWS_ACCESS_KEY_ID;
    global $AWS_SECRET_ACCESS_KEY;
    global $post;
    $STRING = '';
}

```

```

$ID = $post->ID;

$stat = 'publish';
$tax = 'post_tag';
$url = get_option('siteurl');

$sql = "select c.name as tag
      from $wpdb->posts a, $wpdb->term_relationships b, $wpdb->terms c,
      $wpdb->term_taxonomy d
      where a.ID = b.object_id
            and b.term_taxonomy_id = c.term_id
            and c.term_id = d.term_id
            and d.taxonomy = '$tax'
            and a.post_status='$stat'
            and a.ID = $ID";
$tags = $wpdb->get_results($sql);

foreach ((array) $tags as $t){
    $TAGS[] = $t->tag;
}

```

The next part of the function deals with instantiating an Amazon SimpleDB Client and processing the tags extracted from the database. Notice that you're taking the time to rewrite the tags while you're busy building the SimpleDB query. What you want out of the query is to take each tag and string together a series of space-delimited tags:

```

require_once 'Amazon/SimpleDB/Client.php';
$service = new Amazon_SimpleDB_Client($AWS_ACCESS_KEY_ID, $AWS_SECRET_

    $ctr = 0;
foreach ($TAGS as $tag){
    if ($ctr == 0){
        $query = "'tag' = '$tag'";
    }else{
        $query .= " or 'tag' = '$tag'";
    }
    $STRING .= "

```

The last part of the exercise is the most complicated. You are going to run your query (making sure to ignore any item with the current post's ID) and then process the query results such that you end up with a bulleted list of related content, one link per list item.

```

require_once('Amazon/SimpleDB/Model/GetAttributes.php');
$attribute = new Amazon_SimpleDB_Model_GetAttributes();
$attribute->setDomainName($domain);

$queryArray = array ("DomainName" => $domain, "QueryExpression" =>

try {
$response = $service->query($queryArray);

if ($response->isSetQueryResult()) {
$queryResult = $response->getQueryResult();
$itemNameList = $queryResult->getItemName();

$STRING .= "<ul>";
    foreach ($itemNameList as $itemName) {
        $attribute->setItemName($itemName);
        $attrib_response = $service->getAttributes($attribute);
        $STRING .= "<li><a href='$itemName'>";
            if ($attrib_response->isSetGetAttributesResult()) {
                $getAttributesResult = $attrib_response->getGetAttributesResult();
                $attributeList = $getAttributesResult->getAttribute();
                foreach ($attributeList as $ATT) {
                    if($ATT->getName() == "title"){
                        $STRING .= $ATT->getValue();
                    }
                }
            }
        $STRING .= "</a></li>";
    }
    $STRING .= "</ul>";
}
}

```

```

return $STRING;

} catch (Amazon_SimpleDB_Exception $ex) {

echo("Caught Exception: " . $ex->getMessage() . "\n");
echo("Response Status Code: " . $ex->getStatusCode() . "\n");
echo("Error Code: " . $ex->getErrorCode() . "\n");
echo("Error Type: " . $ex->getErrorType() . "\n");
echo("Box Usage: " . $ex->getBoxUsage() . "\n");
echo("Request ID: " . $ex->getRequestId() . "\n");
echo("XML: " . $ex->getXML() . "\n");
}

}

```

Finally, to make everything work, you add an action to the plugin:

```
add_action('the_tags','sdb_related_content');
```

This action prints out your new tags and a list of related content where the tags normally print.

Creating an Options Page

Now that you're storing tags in Amazon SimpleDB and retrieving that same data to build related links, it's time to integrate the plugin with the WordPress admin area. Fortunately, that isn't as complicated as it sounds.

In the following example, you'll build an Options page that lets you manage your Amazon access key, secret password, and Amazon SimpleDB domain. These three pieces of information are crucial to your plugin working properly.

The first thing you're going to do is create a function called `modify_menu_sdb()` in your plugin. This function creates an options page that calls another function called `sdb_options()`, which you'll create in a few minutes.

```

function modify_menu_sdb () {
    add_options_page(
        'SimpleDB Tagging Engine', //Title
        'SimpleDB Tagging Engine', //Sub-menu title
        'manage_options', //Security
        __FILE__, //File to open
        'sdb_options' //Function to call
    );
}

```

The `sdb_options()` function either posts a form (if it detects a form POST) or displays the form.

```

function sdb_options () {
    echo '

```

SimpleDB Options

```

';
    if ($_REQUEST['submit']){
        update_sdb_options();
    }
    print_sdb_form();
    echo '
';
}

```

Lastly, you have the `update_sdb_options()` and `print_sdb_form()` functions. Nothing too exciting here—they do the heavy lifting of storing the information in the database or displaying the form.

```

function update_sdb_options() {
    $updated = false;
    if ($_REQUEST['access_key']) {
        update_option('aws_access_key_id', $_REQUEST['access_key']);
        $updated = true;
    }
    if ($_REQUEST['secret_pw']) {
        update_option('aws_secret_access_key', $_REQUEST['secret_pw']);
        $updated = true;
    }
}

```

```

    }

    if ($_REQUEST['domain']) {
        update_option('aws_simpledb_domain', $_REQUEST['domain']);
        $updated = true;
    }
    if ($updated) {
        echo '<div id="message" class="updated fade">';
        echo '<p>Options Updated</p>';
        echo '</div>';
    } else {
        echo '<div id="message" class="error fade">';
        echo '<p>Unable to update options</p>';
        echo '</div>';
    }
}

function print_sdb_form () {
    $storedKey = get_option('aws_access_key_id');
    $storedPW = get_option('aws_secret_access_key');
    $storedDomain = get_option('aws_simpledb_domain');

    echo "
    <form method='post'>
        <label>Access Key ID:
            <input type='text' name='access_key' value='\$storedKey' si
        </label><br/><br/>
        <label>Secret Access Key:
            <input type='text' name='secret_pw' value='\$storedPW' siz
        </label><br/><br/>
        <label>Domain:
            <input type='text' name='domain' value='\$storedDomain' siz
        </label><br/><br/>
        <input type='submit' name='submit' value='Submit' />
    </form>";
}

```

Only two tasks left. The first is to add an action to associate the `modify_menu_sdb()` function with `admin_menu`. The `admin_menu` action hook is the one hook associated with the administrative menus:

```
add_action('admin_menu', 'modify_menu_sdb');
```

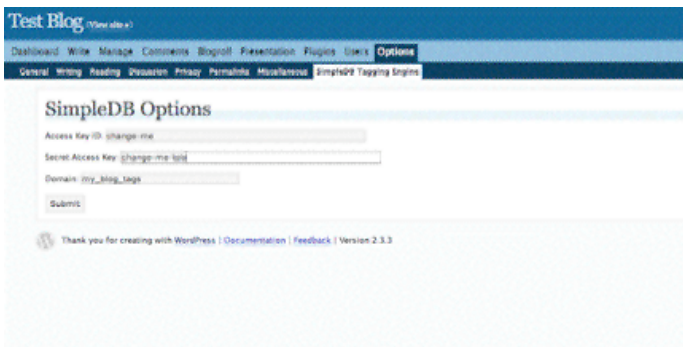
Then, go back to where you hardcoded your access key, secret password, and domain and replace those lines with the following lines of code:

```

$AWS_ACCESS_KEY_ID = get_option('aws_access_key_id');
$AWS_SECRET_ACCESS_KEY = get_option('aws_secret_access_key');
$domain = get_option('aws_simpledb_domain');

```

That's it! You now have a working options page in your WordPress admin area. Your Amazon SimpleDB Tagging Engine is complete.



The Plugin Process in Five Parts

You can create a WordPress plugin that stores tags in Amazon SimpleDB. The process can be broken down into five main parts:

1. Extract tags from the WordPress database tables.
2. Connect to Amazon SimpleDB.

3. Add tag information to SimpleDB, using a unique domain and item names.
4. Retrieve that information to build related content lists.
5. Manage plugin options through an options page.

Learning More About AWS

This article highlights a few aspects of working with AWS. Here are a few more resources available to PHP developers to help you learn more.

Common Resources on AWS

- [AWS web site](#) - Learn more about each web service on the AWS web site.
- [Developer Connection web site](#) - The community web site for AWS developers includes forums on AWS, a Solutions Catalog for examples of what your peers have built, and more.
- [Resource Center](#) - Part of the Developer Connection web site, the Resource Center has links to tutorials, code samples, technical documentation, and other resources for building your application on AWS.

Great Resources for PHP and Amazon SimpleDB Developers

- Find more at the [Amazon SimpleDB Limited Beta](#) site.
- Find technical documentation at [AWS Developer Connection Resources for Amazon SimpleDB](#).
- Use the [Javascript Scratchpad for Amazon SimpleDB](#) to test different functions.

About the Author

Thomas Myer is owner of Austin-based Triple Dog Dare Media (<http://www.tripledogs.com>). He is a Web developer, author of *Lead Generation on the Web* (O'Reilly) and *No Nonsense XML Web Development with PHP* (SitePoint), a consultant, and speaker. He is currently hard at work on a CodeIgniter book for WROX.

Related Documents

 [Amazon SimpleDB Tagging Plugin](#)

Discussion

 [Create a New Discussion](#)

No discussion has been created for this document.

Reviews

 [Write a Review](#)

Be the first to [review this](#).

Have something you'd like to see here? [Give us your feedback](#) on our Developer Connection Feedback forum.

[Conditions of Use](#) | [Privacy Notice](#) © 2006-2008 Amazon Web Services LLC or its affiliates. All rights reserved.