

# Cultured Perl: Perl and the Amazon cloud, Part 3

## Upload images and create, edit, and delete comments

Skill Level: Intermediate

[Teodor Zlatanov \(tzz@lifelogs.com\)](mailto:tzz@lifelogs.com)

Programmer

Gold Software Systems

14 Jun 2009

This [five-part series](#) walks you through building a simple photo-sharing Web site using Perl and Apache to access Amazon's Simple Storage Service (S3) and SimpleDB. In this installment, follow your site's interaction with SimpleDB by learning how the URL creates a SimpleDB record for the uploaded file. Also learn how to create, edit, and delete comments as SimpleDB records on a photo for a particular user.

It's been awhile since my last installment, so here's the story so far:

- [Part 1](#) explained the S3/SimpleDB architectures and how to use them through practical examples.
- [Part 2](#) showed how to upload a file into S3 from a Web page through an HTML form, minimizing the load on the server.

### To get the most from this series

This series requires beginner-level knowledge of HTTP and HTML, as well as intermediate-level knowledge of JavaScript and Perl (inside an Apache `mod_perl` process). Some knowledge of relational databases, disk storage, and networking will be helpful. The series gets increasingly technical, so see the [Resources](#) section if you need help with any of those topics.

Now we'll really start diving into the photo-sharing site's interaction with Amazon's SimpleDB by learning how the successful URL creates a SimpleDB record for the

uploaded file and learning how to create, edit, and delete comments as SimpleDB records on a photo for a particular user. (Remember, we are not comparing SimpleDB to Google's BigTable or to standalone solutions like CouchDB.)

As in previous installments, I use share.lifelogs.com as the domain name.

## But first, the database structure

Referring back to [Part 1](#), we set up a table structure like the one shown in Listing 1:

### Listing 1. The table structure from Part 1

```
share_photos:
"http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif"
{ user: "ted", name: "Amazon Logo" }

"http://images.share.lifelogs.com/funny.jpg"
{ user: "bob", name: "Funny Picture", s3bucket: "images.share.lifelogs.com" }

share_users:
"ted" { given: "Ted", family: "Zlatanov" }
"bob" { given: "Bob", family: "Leech" }

share_comments:
"random-string"
{
  url: "http://images.share.lifelogs.com/funny.jpg",
  comment: "Ha ha",
  posted_when: "2009-03-01T19:00:00+05"
}

"random-string2"
{
  user: "ted",
  url: "http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif",
  comment: "No it doesn't",
  posted_when: "2009-03-01T20:00:01+05"
}

"random-string3"
{
  url: "http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif",
  comment: "No it doesn't",
  reply_to: "random-string2",
  posted_when: "2009-03-01T20:00:01+05"
}
```

In this version of the structure, you can store all the comments in a thread inside one comment structure, but that would make it dangerous to allow two users to delete or edit a comment in the same thread at the same time.

As I noted in Part 1, this structure is subject to change. One big advantage of SimpleDB is its flexibility, so let's use that. "A foolish consistency is the hobgoblin of little minds," as Emerson said. Of course, he also said "I hate quotations." Damn those waffling transcendentalists.

I found the SimpleDB scratchpad less useful than making the calls directly from code, but perhaps you'll like it. Look in the [Resources](#) section for the scratchpad URL. You can also purchase several SimpleDB management tools, use the free one for Firefox, or try out the typical shell. It's all in the [Resources](#) section; go forth and manage.

## Uploading and modifying images

First of all, note that SimpleDB allows multiple values for a single key. So the image URL I mentioned or the comment text could be arrays instead of single values. We won't use this feature, preferring instead to keep things simple with one value per key.

Let's start with the image update. Every time an upload is done, you'll run code to add the new image to the `share_photos` table. [Part 2](#) showed you the S3 upload form; and next time we'll connect that form to the code written here.

For now, let's write a simple script to add an image. You're given a user name, a URL, an image name, and an optional S3 bucket name. The S3 bucket will be just a field in the table; the URL will be sufficient to display and use the image. We'd like to reject uploads with a duplicate URL. But can we?

The problem with a distributed data store is that the data you just put into it may not have made it out to the edge of the network. It's like calling Europe from Japan (or doing anything where you experience the latency of the network or voice signal): there's a slight delay after every sentence while the other side synchronizes with you. You can start talking without pause, but then the other side's answers will start overlapping with your words and you'll feel like a Real Manager In An Important Meeting. So while latency is good for a *forceful* discourse, it's not so nice if you want a civilized conversation.

Similarly, when you put data into the SimpleDB system, there's a pause while the data flows out to a data center. Now, this part has not been confirmed by Amazon, but I've been told it becomes part of galactic consciousness and improves life for slug-like beings in the Runu system (three parsecs off Betelgeuse, then hang a left and keep going until you see it). So you're doing a favor for slug-like beings in the Runu system every time you use SimpleDB. And you thought this article was not *educational*.

Anyway, in parallel with improving galactic consciousness, your data goes live and then your queries will see it. But if you made queries before all this happened, not only will galactic consciousness not improve, you'll get old data. So it's not as simple as working with a classic database like DB2 in which your data goes live and there's ACID ensuring your transactions are committed when the database says they are. With SimpleDB and other "eventually consistent" databases, you just have to live

with the uncertainty.

The point is, updating the images is not so simple. We'd like to reject uploads with a duplicate URL, but that's not always possible. Imagine Alice uploads `http://horsey.com/wilbur.png` and Bob also uploads `http://horsey.com/wilbur.png` at the same time. If Alice's upload goes in first and Bob doesn't see it, Bob's upload will overwrite it. So what do we do?

First of all, you might ask, what's the harm? Users are inconvenienced but it's not a huge deal. Also, it's not a terribly likely occurrence. Well, we want happy users and, if we are obsessive about quality, it will bother us to our death bed to release something so obviously broken.

Rather than carrying unhappiness to the grave, we'll modify our table design for images, as shown in Listing 2:

### Listing 2. The modified table design

```
share_photos:

"random-string10"
{ url: "http://developer.amazonwebservices.com/connect/images/amazon/logo_aws.gif",
  user: "ted", name: "Amazon Logo" }

"random-string11"
{ url: "http://images.share.lifelogs.com/funny.jpg", user: "bob", name: "Funny Picture",
  s3bucket: "images.share.lifelogs.com" }
```

The `random-strings` you've seen so far will be UUIDs. It's not perfect, but at least our images won't collide if the URLs are the same. But wait...what about image comments? Easy; we'll just change the foreign key as shown in Listing 3:

### Listing 3. Changing the foreign key

```
share_comments:

"random-string3"
{
  image_id: "random-string10",
  comment: "No it doesn't",
  reply_to: "random-string2",
  posted_when: "2009-03-01T20:00:01+05"
}
```

Now we have to be aware that there may be multiple entries in `share_photos` with the same URL, but otherwise the system seems to be okay.

Understand that rather than showing you the artificially ready final version of the tables, we're working through them together. This lets me showcase the flexibility of SimpleDB and also showcase agile development at its best: invent, test, refine,

repeat. Rather than planning *everything*, we plan just enough to get us to the next stage, though:

- We don't forget about the larger picture at any time.
- Architectural decisions are not made or changed as lightly as task-specific decisions.

So an image upload is simple, right? Just add a new entry to SimpleDB with the given URL, image name, and user name. A S3 bucket is optional. This is done with a `PutAttributes` call.

Modifying an image is equally easy, but let's just allow changing the name for now. This is also done with `PutAttributes`.

## Adding and modifying comments

Refer to the previous section for the `share_comments` table. Simple stuff, really: Adding a comment will require the comment text, image ID, and optionally the parent comment ID and a user name. Modifying a comment will allow changes only to the comment text for now.

## The standalone script

I've included a standalone Perl script (`simple_go.pl`; you can get it from the [Download](#) section at the bottom of this article) to do the tasks I've listed (add and modify image, add and modify comment). It won't create the domains, so you'll need to create the SimpleDB domains `share_photos.share.lifelogs.com` and `share_comments.share.lifelogs.com` externally. This is trivial using any of the SimpleDB management tools. Note that the `--domain` switch will change `share.lifelogs.com` to something else for the full domain name (stored in `$full_domain`).

The script uses the CPAN `Data::UUID` module to generate new unique identifiers.

The script is pretty cavalier about handling errors, choosing to `die()` at every opportunity. This is lazy and despicable, and you should not do it unless you're writing articles and want to show the world the utter depravity of programmers that write articles.

You're welcome.

The last tasks are to submit a `SELECT` statement and to delete an item. I'll show you how to implement them here because they are simple and you'll need them later.

To list images, you call the script as shown in Listing 4:

#### Listing 4. Listing images

```
./simple_go.pl -l -i --ak=accesskey --sk=secretkey
```

A word of caution: make sure the machine you run this on is not used by others. They can look at the list of processes and see your Amazon secret key. Similarly, if you are in a shell that keeps history, your secret key will end up in your history file. A better approach is to pass a file name and get the password from that file, but in the interest of simplicity, I did not implement it.

To list comments:

#### Listing 5. Listing comments

```
./simple_go.pl -l -c --ak=accesskey --sk=secretkey
```

Pretty simple so far. Internally the script calls the `$service->select()` method, parses the results, and prints the data with `show_list()`. Everything is done with the assumption that a key has just one value (note we specify `Replace=true` in the `put()` method), so this is not a general-purpose SimpleDB script.

Why not a general-purpose script? We don't need it. Let's get a simple solution working now. If we need multiple values, we can adapt the script later or just write new code. This script is the playground for creating our Web site's database structures.

Don't be tempted to use this script in the real site either ("I'll just call `system()` and let the errors go to a log file"). Yes, it's a few hundred lines of code and it works, but every prototype must be discarded without regrets so a real program can be written. Let's not make an exception for this one even if we've grown somewhat attached to its one-space indents and haphazard (er, "creative") layout.

Back to the script. Creating a new image is simple (bucket is optional):

#### Listing 6. Creating a new image

```
./simple_go.pl -i --ak=accesskey --sk=secretkey -u ted --url="any url"  
--name="any name you like" --bucket=mybucket
```

Editing the image name (`-l -i` gave us an ID of 25EC17B8-0F6B-11DE-A1A1-944E07F9DEC1). This now creates an image with a unique UUID:

## Listing 7. Creating a new image with a unique UUID

```
./simple_go.pl -i --ak=accesskey --sk=secretkey --name="new name"  
--id=25EC17B8-0F6B-11DE-A1A1-944E07F9DEC1
```

Similarly, creating a comment is easy (`user` and `refcommentid` are optional):

## Listing 8. Creating a comment

```
./simple_go.pl -c --ak=accesskey --sk=secretkey -u ted --refimageid="any image ID"  
--text="the text" --refcommentid='any comment ID'
```

Again, the comment ID will be a unique UUID. Editing a comment's text is done like so (`-l -c` gave us an ID of `4BE2EA0A-0F6B-11DE-976B-A542FC6BD07C`):

## Listing 9. Creating a comment with a unique UUID

```
./simple_go.pl -c --ak=accesskey --sk=secretkey --text="the text"  
--id=4BE2EA0A-0F6B-11DE-976B-A542FC6BD07C
```

Finally, deleting images or comments works like so:

## Listing 10. Deleting images and comments

```
./simple_go.pl --delete -i --ak=accesskey --sk=secretkey  
--id=25EC17B8-0F6B-11DE-A1A1-944E07F9DEC1  
./simple_go.pl --delete -c --ak=accesskey --sk=secretkey  
--id=4BE2EA0A-0F6B-11DE-976B-A542FC6BD07C
```

## Wrapup

This installment showed how images and comments are created, edited, and deleted in the SimpleDB database that backs the photo-sharing site we're building.

We established the schema (loosely) and implemented a tool to add, list, modify, and delete images and comments. We settled on UUIDs as the primary key for both images and comments to prevent the unlikely scenario of two users uploading the same image URL at the same time.

We also established that we would use only a single value per key since currently our schema does not require more and we're aiming for simplicity. This shortcoming may need to be addressed later in code, but for now we will live with it.

In Part 4, you see how to start putting it all together in a `mod_perl` Web site.



## Downloads

Description	Name	Size	Download method
Sample script for this article	simple_go.zip	4KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- Read "[Cultured Perl: Perl and the Amazon cloud, Part 1](#)" (developerWorks, March 2009) to understand the benefits and drawbacks of using Amazon's S3 and SimpleDB for Web site building. "[Cultured Perl: Perl and the Amazon cloud, Part 2](#)" (April 2009) shows how to upload a file into S3 from a Web page through an HTML form, minimizing the load on the server.
- Learn everything there is to learn about the amazing [JavaScript MimeType](#), a property of the navigator object, a top-level object that is the object representation of the client Internet browser or Web navigator program that is being used.
- CloudBerry Lab offers a [roundup of SimpleDB management tools for various platforms](#).
- [mod\\_perl](#) is a Perl interpreter embedded in Apache.
- [Prototype](#) is a JavaScript framework to make it easier to develop dynamic Web applications via a toolkit for class-driven development and the "nicest" Ajax library on Earth. And here's an excellent article on using it, "[Developer Notes for prototype.js](#)" by Sergio Pereira.
- Yes, Virginia, it's a Web-based service and therefore can suffer [outages](#); an important consideration.
- In the [developerWorks Linux zone](#), find more resources for Linux developers, and scan our [most popular articles and tutorials](#).
- See all [Linux tips](#) and [Linux tutorials](#) on developerWorks.
- Stay current with [developerWorks technical events and Webcasts](#).

## Get products and technologies

- The [Javascript Scratchpad for Amazon SimpleDB](#) is a simple HTML and JavaScript application that allows you to explore the Amazon SimpleDB API without writing any code.
- The Google [typica shell](#) is a simple, thread-safe API to access Amazon's SQS, EC2, SimpleDB, and DevPay LS Web services that uses the QUERY interfaces and patterns the methods after those available from Amazon in their SOAP client for these services.
- The [Open Source Simple DB Firefox Plugin: SDB Tool](#) is a free Firefox-based SimpleDB manager.
- This [SimpleDB module](#) simplifies Amazon::SimpleDB usage. I took some ideas from it.

- Get the technologies du jour (includes pricing):
  - [Amazon S3](#)
  - [Amazon SimpleDB](#)
- At [CPAN \(the Comprehensive Perl Archive Network\)](#) site, you can find scads of modules. And module documentation.
- Get the lovely S3 management-utility add-on for Firefox, [S3Fox](#), the [Amazon S3 Firefox Organizer](#).
- With [IBM trial software](#), available for download directly from developerWorks, build your next development project on Linux.

### Discuss

- Get involved in the [My developerWorks community](#); with your personal profile and custom home page, you can tailor developerWorks to your interests and interact with other developerWorks users.

## About the author

Teodor Zlatanov

Teodor Zlatanov emerged with an M.S. in computer engineering from Boston University in 1999. He has worked as a programmer since 1992, using Perl, Java, C, and C++. His interests are in open source work on text parsing, database architectures, user interfaces, and UNIX system administration.