Amazon SimpleDB

Developer Guide

API Version 2009-04-15

Amazon SimpleDB: Developer Guide

Table of Contents

Welcome to Amazon SimpleDB 1
Introduction to Amazon SimpleDB 2
Features 2
How Amazon Charges for Amazon SimpleDB 3
Storage 3
Data Transfer 3
Machine Utilization 4
Viewing Your Bill 4
Amazon SimpleDB Concepts 5
Data Model 5
Operations 6
API Summary 7
Consistency 7
Concurrent Applications 7
Limits 10
Data Set Partitioning 11
AWS Identity and Access Management 12
Using Amazon SimpleDB 13
Available Libraries 13
Making API Requests 13
Region Endpoints 14
Making REST Requests 14
Request Authentication 17
What Is Authentication? 18
Creating an AWS Account 18
Managing Users of Amazon SimpleDB 20
Using Temporary Security Credentials 22
HMAC-SHA Signature 23
Working with Domains 29
Creating a Domain 29
Verifying the Domain 30
Deleting a Domain 30
Working with Data 31
Putting Data into a Domain 31
Getting Data from a Domain 32
Deleting Data from a Domain 32
Conditionally Putting and Deleting Data 33
Performing a Conditional Put 33
Performing a Conditional Delete 35
Using Select to Create Amazon SimpleDB Queries 36
Comparison Operators 37
Sample Query Data Set 39
Simple Queries 40
Range Queries 40
Queries on Attributes with Multiple Values 41
Multiple Attribute Queries 42
Sort 43
Count 44
Select Quoting Rules 44
Working with Numerical Data 45
Negative Numbers Offsets 45
Zero Padding 46
Dates 46
Tuning Queries 47

Tuning Your Queries Using Composite Attributes 47
Data Set Partitioning 48
Working with XML-Restricted Characters 49
API Reference 50
API Usage 50
API Conventions 50
WSDL Location and API Version 50
API Error Retries 51
Common Parameters 52
Request Parameters 52
Request Parameter Formats 53
Common Response Elements 54
Common Error Responses 54
Operations 54
BatchDeleteAttributes 54
BatchPutAttributes 57
CreateDomain 62
DeleteAttributes 63
DeleteDomain 68
DomainMetadata 70
GetAttributes 72
ListDomains 75
PutAttributes 76
Select 81
API Error Codes 85
About Response Code 503 85
Amazon SimpleDB Error Codes 85
Amazon SimpleDB Glossary 90
Document History 92

Welcome to Amazon SimpleDB

This is the Developer Guide for *Amazon SimpleDB*. This guide provides a conceptual overview of Amazon SimpleDB, programming reference material, and a detailed API reference.

Amazon SimpleDB is a web service for running queries on structured data in real time. This service works in close conjunction with Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2), collectively providing the ability to store, process and query data sets in the cloud. These services are designed to make web-scale computing easier and more cost-effective for developers.

How Do I...?	Relevant Sections
Learn if Amazon SimpleDB is right for my use case	Amazon SimpleDB Detail Page
Learn about the Amazon SimpleDB data model	Data Model (p. 5)
Learn how to tune Amazon SimpleDB queries	Tuning Queries (p. 47)
Find information about Amazon SimpleDB operations	API Reference (p. 50)
Find and use different Amazon SimpleDB endpoints	Region Endpoints (p. 14)
Find information about Amazon SimpleDB libraries	Amazon SimpleDB Sample Code and Libraries
Get help from other developers	Amazon SimpleDB Forums

Introduction to Amazon SimpleDB

Topics

- Features (p. 2)
- How Amazon Charges for Amazon SimpleDB (p. 3)
- Viewing Your Bill (p. 4)

This introduction to Amazon SimpleDB is intended to give you a detailed summary of this web service. After reading this section, you should have a good idea of what it offers and how it can fit in with your business.

Traditionally, the type of functionality provided by Amazon SimpleDB has been accomplished with a clustered relational database that requires a sizable upfront investment, brings more complexity than is typically needed, and often requires a DBA to maintain and administer. In contrast, Amazon SimpleDB is easy to use and provides the core functionality of a database - real-time lookup and simple querying of structured data - without the operational complexity. Amazon SimpleDB requires no schema, automatically indexes your data and provides a simple API for storage and access. This eliminates the administrative burden of data modeling, index maintenance, and performance tuning. Developers gain access to this functionality within Amazon's proven computing environment, are able to scale instantly, and pay only for what they use.

Features

Following are some of the major Amazon SimpleDB attributes:

Features

- **Simple to use—**Amazon SimpleDB provides streamlined access to the lookup and query functions that traditionally are achieved using a relational database cluster while leaving out other complex, often-unused database operations.

  The service allows you to quickly add data and easily retrieve or edit that data through a simple set of API calls. Accessing these capabilities through a web service also eliminates the complexity of maintaining and scaling these operations
- **Flexible—**With Amazon SimpleDB, it is not necessary to pre-define all of the data formats you will need to store; simply add new attributes to your Amazon SimpleDB data set when needed, and the system will automatically index your data accordingly.

  The ability to store structured data without first defining a schema provides developers with greater flexibility when building applications.
- **Scalable—**Amazon SimpleDB allows you to easily scale your application. You can quickly create new domains as your data grows or your request throughput increases.

  Currently, you can store up to 10 GB per domain and you can create up to 250 domains.
- **Fast—**Amazon SimpleDB provides quick, efficient storage and retrieval of your data to support high performance web applications.
- **Reliable—**The service runs within Amazon's high-availability data centers to provide strong and consistent performance.

  To prevent data from being lost or becoming unavailable, your fully indexed data is stored redundantly across multiple servers and data centers.

- **Designed for use with other Amazon Web Services—**Amazon SimpleDB is designed to integrate easily with other web-scale services such as Amazon EC2 and Amazon S3.

  For example, developers can run their applications in Amazon EC2 and store their data objects in Amazon S3. Amazon SimpleDB can then be used to query the object metadata from within the application in Amazon EC2 and return pointers to the objects stored in Amazon S3.
- **Inexpensive—**Amazon SimpleDB passes on to you the financial benefits of Amazon's scale. You pay only for resources you actually consume.

  Compare this with the significant up-front expenditures traditionally required to obtain software licenses and purchase and maintain hardware, either in-house or hosted. This frees you from many of the complexities of capacity planning, transforms large capital expenditures into much smaller operating costs, and eliminates the need to over-buy "safety net" capacity to handle periodic traffic spikes.

How Amazon Charges for Amazon SimpleDB

Amazon SimpleDB pricing is based on your actual usage. Your usage is measured and rounded up to the nearest cent.

Amazon SimpleDB charges you for the following types of usage:

- **Structured Data Storage—**Measures the size of your billable data by adding the raw byte size of the data you upload + 45 bytes of overhead for each item, attribute name and attribute-value pair.
- **Data Transfer—**Measures the amount of data transferred for every operation.

  Data transferred between Amazon SimpleDB and other Amazon Web Services (e.g., Amazon S3, Amazon EC2, Amazon SQS, and others) is free of charge.
- **Machine Utilization—**Measures the *machine utilization* of each request and charges based on the amount of machine capacity used to complete the particular request (SELECT, GET, PUT, etc.).

> **Note**
> Amazon Web Services provides a free tier of Amazon SimpleDB usage. The free tier is a monthly offer. Free usage does not accumulate.
> Any data stored as part of the free tier program must be actively used. If a domain is not accessed for a period of 6 months, it will be subject to removal at the discretion of Amazon Web Services.

Storage

You are charged for the amount of storage your data uses each month which can be considered an average of the month. For example, if you use one gigabyte for the month, you are charged for one gigabyte of storage. If you use zero gigabytes for the first half of the month and two gigabytes for the second half, you are also charged for one gigabyte of storage.

Several times each day, Amazon SimpleDB measures the amount of storage used by all of the objects in your account. Amazon SimpleDB stores this information in byte-hours and averages it with all other recorded measurements at the end of the billing cycle.

Data Transfer

Amazon charges you for the amount of data transferred into and out of Amazon SimpleDB. For every operation, Amazon SimpleDB monitors the amount of data sent and received and records the data. Once

per hour, the usage total is recorded to your account. This information is stored and totaled at the end of the billing cycle.

Machine Utilization

For each successful request, Amazon SimpleDB charges you for the amount of machine capacity used to complete that request.

You are also charged for any request that fails with an HTTP 4xx error. For example, if Amazon SimpleDB cannot authorize a request, you will receive an HTTP 403 error (Forbidden) and incur a machine utilization charge for the failed request.

Viewing Your Bill

You can view the charges for your current billing period at any time by going to the AWS Portal.

To view your activity

1. Log in to your AWS account.
2. Move the pointer over **Your Web Services Account**.
3. Click **Account Activity**.

   A list of services to which you subscribe appears.
4. Locate the Amazon SimpleDB service.

Billing Component	Description
View/Edit Service Button	Enables you to view or change settings associated with the Amazon SimpleDB service.
Machine Hour Usage	Shows the machine hour usage cost, current number of hours consumed, and billing for the current cycle.
Data Transferred	Shows the data transfer cost, current amount of data transferred, and billing for the current cycle.
Storage	Shows the storage usage cost, average amount of storage consumed, and billing for the current cycle.
Usage Report	Shows detailed data used to calculate your bill.

Amazon SimpleDB Concepts

Topics

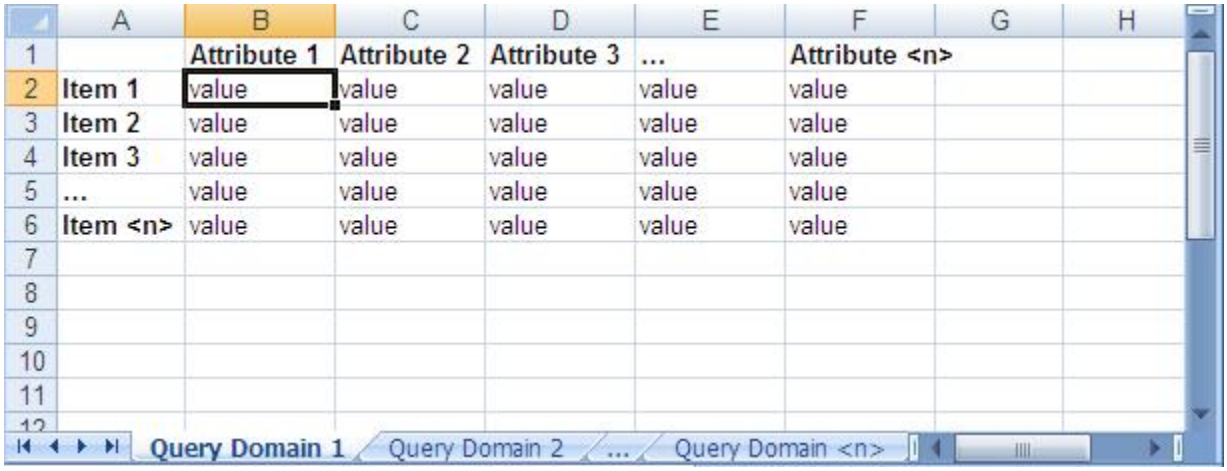
- Data Model (p. 5)
- Operations (p. 6)
- API Summary (p. 7)
- Consistency (p. 7)
- Limits (p. 10)
- Data Set Partitioning (p. 11)
- AWS Identity and Access Management (p. 12)

This section describes the key concepts that you should understand before using Amazon SimpleDB.

Data Model

When using Amazon SimpleDB, you organize your structured data in domains within which you can put data, get data, or run queries.

Domains consist of items which are described by *attribute* name-value pairs. Consider the spreadsheet model shown in the following image.



The components correspond to each part of a spreadsheet:

- **Customer Account—**Represented by the entire spreadsheet, it refers to the Amazon Web Services *account* to which all domains are assigned.
- **Domains—**Represented by the domain worksheet tabs at the bottom of the spreadsheet, domains are similar to tables that contain similar data.

  You can execute queries against a domain, but cannot execute queries across different domains.

- **Items—**Represented by rows, items represent individual objects that contain one or more attribute name-value pairs.
- **Attributes—**Represented by columns, attributes represent categories of data that can be assigned to items.
- **Values—**Represented by cells, values represent instances of attributes for items. An attribute can have multiple values.

Unlike a spreadsheet, however, multiple values can be associated with a cell. For example, an item can have both the color value *red* and *blue*. Additionally, Amazon SimpleDB does not require the presence of specific attributes. You can create a single domain that contains completely different product types. For example, the following table contains clothing, automotive parts, and motorcycle parts.

ID	Category	Subcat.	Name	Color	Size	Make	Model
Item_01	Clothes	Sweater	Cathair Sweater	Siamese	Small, Medium, Large		
Item_02	Clothes	Pants	Designer Jeans	Paisley Acid Wash	30x32, 32x32, 32x34		
Item_03	Clothes	Pants	Sweatpants	Blue, Yellow, Pink	Large		
Item_04	Car Parts	Engine	Turbos			Audi	S4
Item_05	Car Parts	Emissions	02 Sensor			Audi	S4
Item_06	Motorcycle Parts	Bodywork	Fender Eliminator	Blue		Yamaha	R1
Item_07	Motorcycle Parts, Clothing	Clothing	Leather Pants	Black	Small, Medium, Large		

Regardless of how you store your data, Amazon SimpleDB automatically indexes your data for quick and accurate retrieval.

Operations

The following describes components of Amazon SimpleDB operations.

- **Subscriber—**Any application, script, or software making a call to the Amazon SimpleDB service.

  The AWS Access Key ID uniquely identifies each subscriber for billing and metering purposes.
- **Amazon SimpleDB Request—**A single web service API call and its associated data that the subscriber sends to the Amazon SimpleDB service to perform one or more operations.
- **Amazon SimpleDB Response—**The response and any results returned from the Amazon SimpleDB service to the subscriber after processing the request.

  The AWS Platform handles authentication success and failure; failed requests are not sent to the Amazon SimpleDB service.

API Summary

The Amazon SimpleDB service consists of a small group of API calls that provide the core functionality you need to build your application. See Operations (p. 54) in the API Reference chapter for detailed descriptions of each option.

- **CreateDomain—**Create domains to contain your data; you can create up to 250 domains. If you require additional domains, go to https://console.aws.amazon.com/support/home#/case/create?issueType=service-limit-increase&limitType=service-code-simpledb-domains.
- **DeleteDomain—**Delete any of your domains
- **ListDomains—**List all domains within your account
- **PutAttributes—**Add, modify, or remove data within your Amazon SimpleDB domains
- **BatchPutAttributes—**Generate multiple put operations in a single call
- **DeleteAttributes—**Remove items, attributes, or attribute values from your domain
- **BatchDeleteAttributes—**Generate multiple delete operations in a single call
- **GetAttributes—**Retrieve the attributes and values of any item ID that you specify
- **Select—**Query the specified domain using a SQL SELECT expression
- **DomainMetadata—**View information about the domain, such as the creation date, number of items and attributes, and the size of attribute names and values

Consistency

Amazon SimpleDB keeps multiple copies of each domain. A successful write (using PutAttributes, BatchPutAttributes, DeleteAttributes, BatchDeleteAttributes, CreateDomain, or DeleteDomain) guarantees that all copies of the domain will durably persist.

Amazon SimpleDB supports two read consistency options: eventually consistent read and consistent read.

An eventually consistent read (using Select or GetAttributes) might not reflect the results of a recently completed write (using PutAttributes, BatchPutAttributes, DeleteAttributes, or BatchDeleteAttributes). Consistency across all copies of the data is usually reached within a second; repeating a read after a short time should return the updated data.

A consistent read (using Select or GetAttributes with `ConsistentRead=true`) returns a result that reflects all writes that received a successful response prior to the read.

By default, GetAttributes and Select perform an eventually consistent read.

The following table describes the characteristics of eventually consistent read and consistent read.

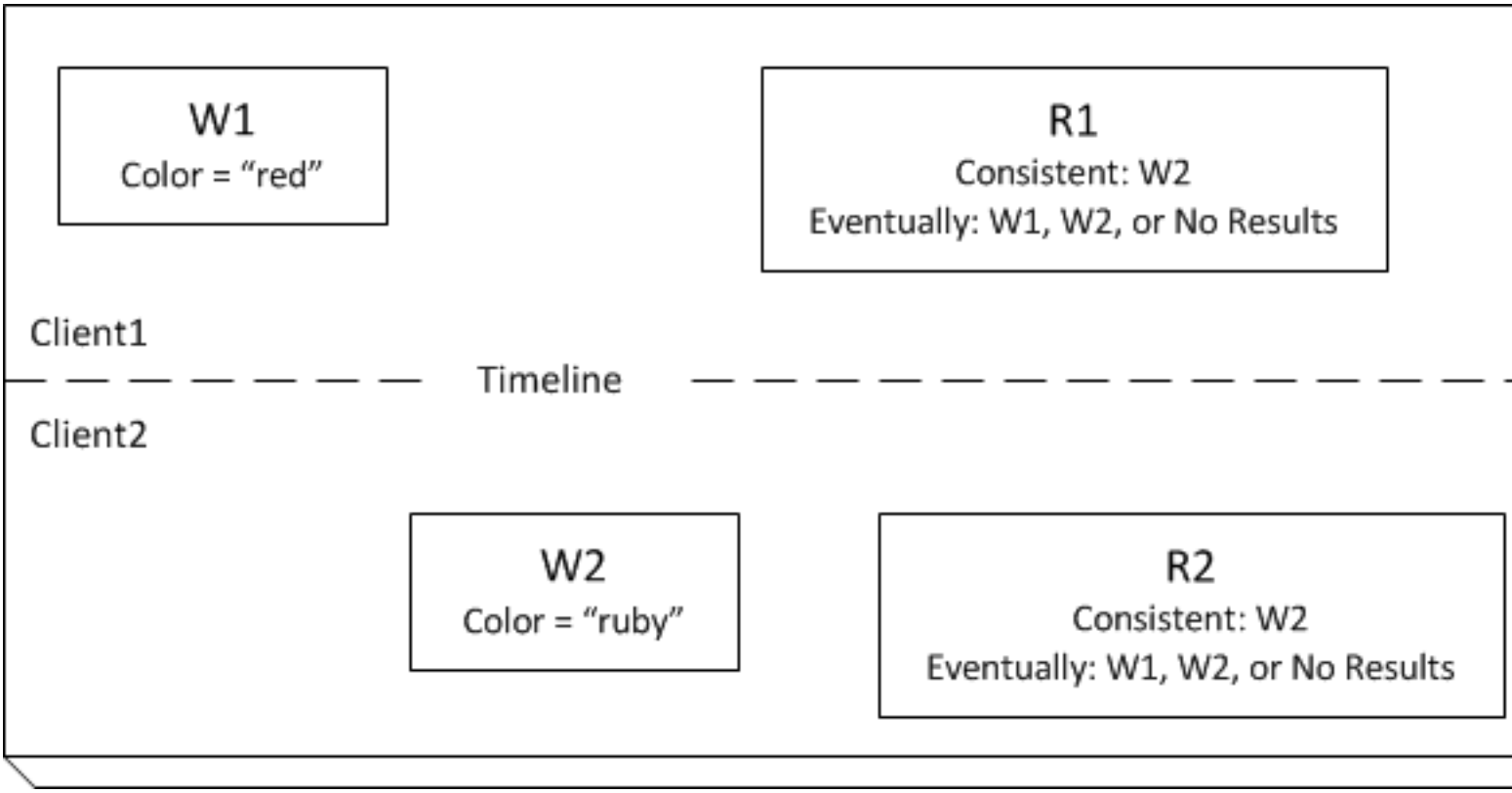
Eventually Consistent Read	Consistent Read
Stale reads possible	No stale reads
Lowest read latency	Potential higher read latency
Highest read throughput	Potential lower read throughput

Concurrent Applications

This section provides examples of eventually consistent and consistent read requests when multiple clients are writing to the same items. Whenever you have multiple clients writing to the same items,

implement some concurrently control mechanism, such as timestamp ordering, to ensure you are getting the data you want.

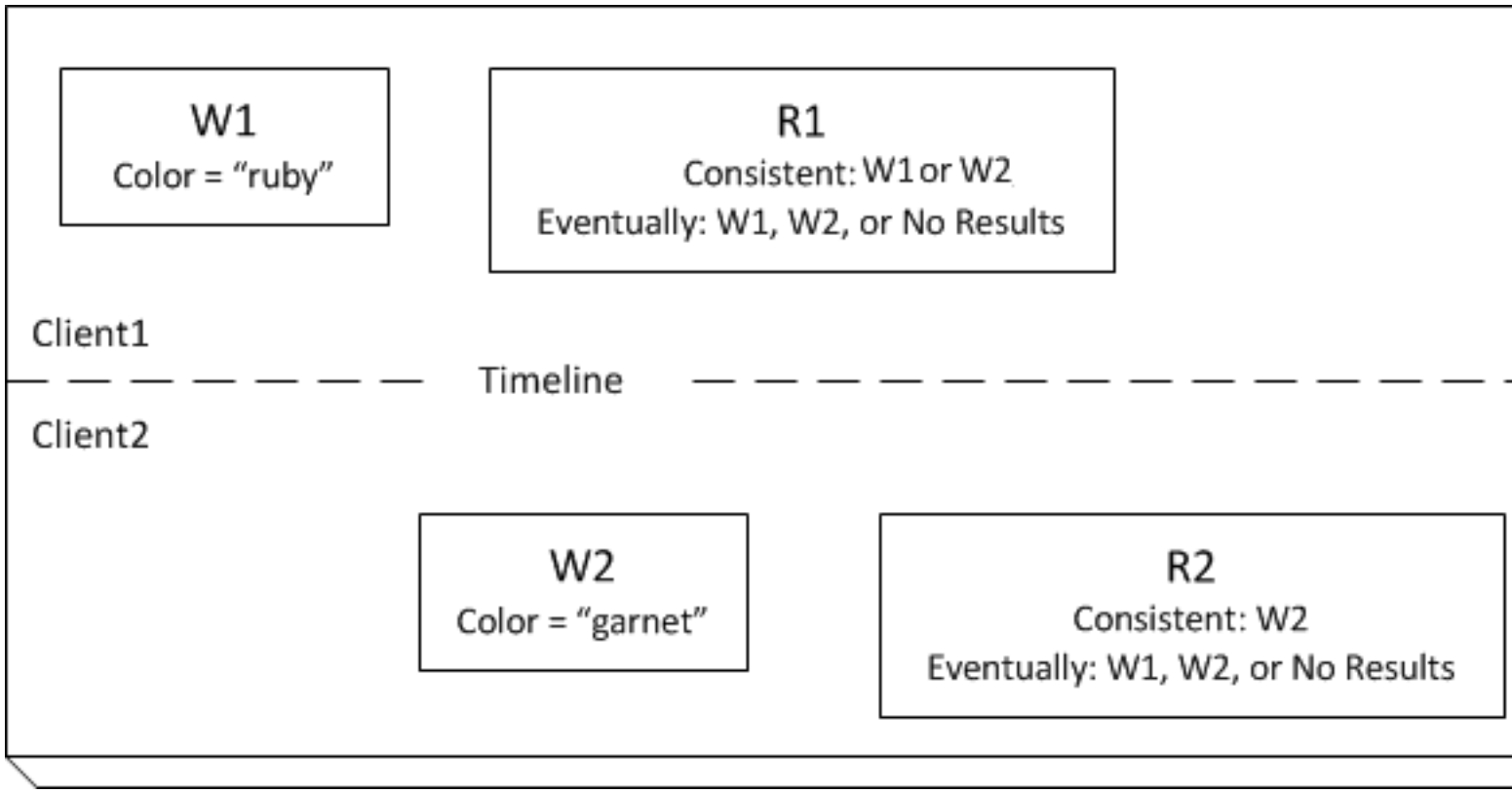
In this example, both W1 (write 1) and W2 (write 2) complete (receive a successful response from the server) before the start of R1 (read 1) and R2 (read 2). For a consistent read, R1 and R2 both return `color = ruby`. For an eventually consistent read, R1 and R2 might return `color = red`, `color = ruby`, or no results, depending on the amount of time that has elapsed.



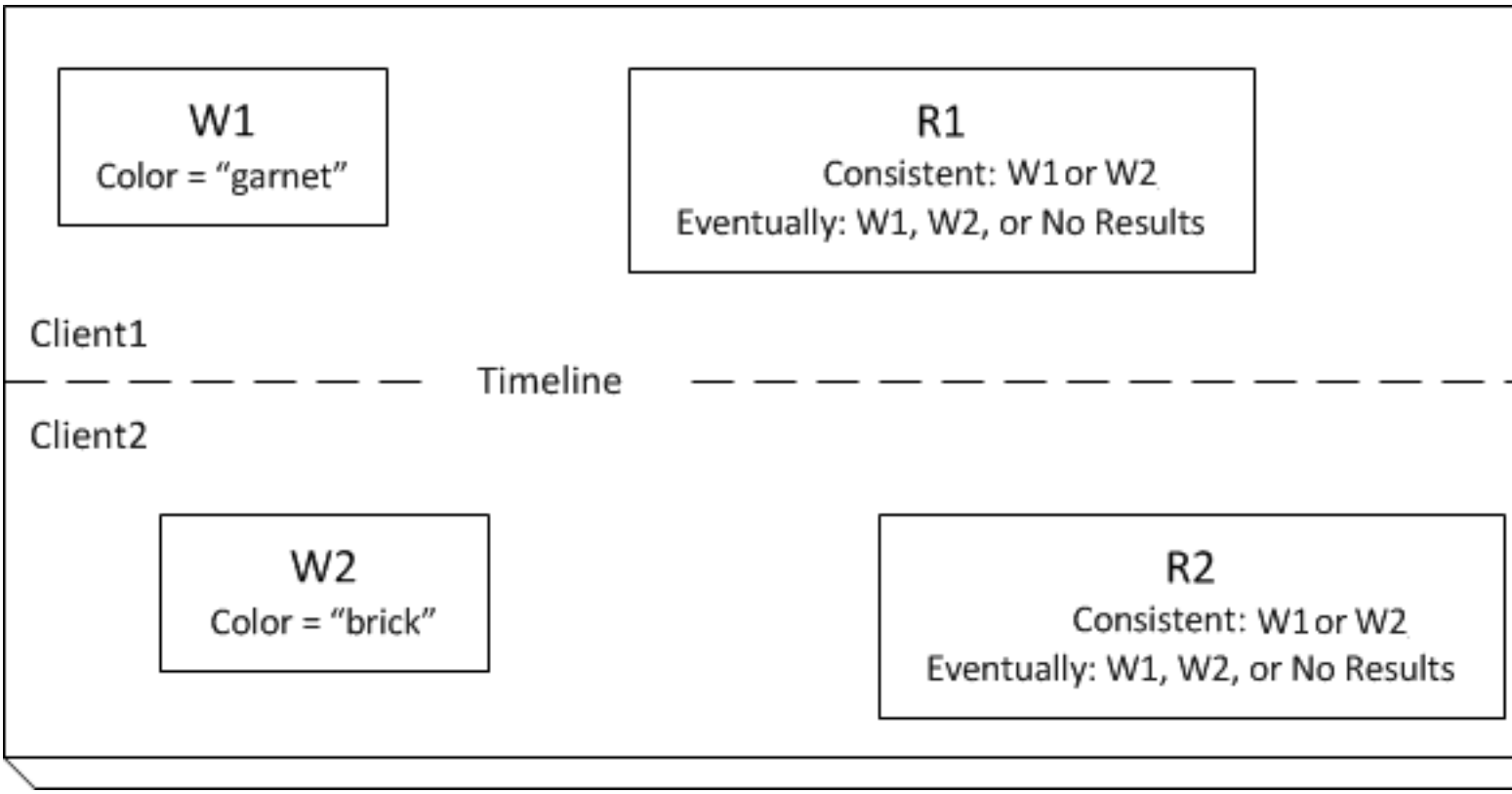
In the next example, W2 does not complete before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet` for either a consistent read or an eventually consistent read. Data is distributed among several servers. If R1 is sent to one server that does not have the W2 values, yet, then R1 returns W1 values. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.

Note
If a failure occurs during the second write operation (W2), the value might change depending on when in the operation the failure occurs.

For a consistent read, R2 returns `color = garnet`. For an eventually consistent read, R2 might return `color = ruby`, `color = garnet`, or no results depending on the amount of time that has elapsed.



In the last example, Client 2 submits W2 before Amazon SimpleDB completes W1, so the outcome of the final value is unknown (`color = garnet` or `color = brick`). Any subsequent reads (consistent read or eventually consistent) might return either value. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.



Limits

Following is a table that describes current limits within Amazon SimpleDB.

Parameter	Restriction
Domain size	10 GB per domain
Domain size	1 billion attributes per domain
Domain name	3-255 characters (a-z, A-Z, 0-9, '_', '-', and '.')
Domains per account	250
Attribute name-value pairs per item	256
Attribute name length	1024 bytes
Attribute value length	1024 bytes
Item name length	1024 bytes
Attribute name, attribute value, and item name allowed characters	All UTF-8 characters that are valid in XML documents.

Parameter	Restriction
	Control characters and any sequences that are not valid in XML are returned Base64-encoded. For more information, see Working with XML-Restricted Characters (p. 49).
Attributes per `PutAttributes` operation	256
Attributes requested per `Select` operation	256
Items per `BatchDeleteAttributes` operation	25
Items per `BatchPutAttributes` operation	25
Maximum items in `Select` response	2500
Maximum query execution time	5 seconds
Maximum number of unique attributes per `Select` expression	20
Maximum number of comparisons per `Select` expression	20
Maximum response size for `Select`	1MB

Data Set Partitioning

Amazon SimpleDB is designed to support highly parallel applications. To improve performance, you can partition your dataset among multiple domains to parallelize queries and have them operate on smaller individual datasets. Although you can only execute a single query against a single domain, you can perform aggregation of the result sets in the application layer. The following is a list of applications that lend themselves to parallelized queries:

- **Natural Partitions—**The data set naturally partitions along some dimension. For example, a product catalog might be partitioned in the "Book", "CD" and "DVD" domains. Although you can store all the product data in a single domain, partitioning can improve overall performance.
- **High Performance Application—**Useful when the application requires higher throughput than a single domain can provide.
- **Large Data Set—**Useful when timeout limits are reached because of the data size or query complexity.

In cases where data sets do not partition easily (e.g., logs, events, web crawler data), you can use hashing algorithms to create a uniform distribution of items among multiple domains.

For example, you can determine the hash of an item name using a well-behaved hash function, such as MD5 and use the last 2 bits of the resulting hash value to place each item in a specified domain.

- If last two bits equal 00, place item in Domain0
- If last two bits equal 01, place item in Domain1
- If last two bits equal 10, place item in Domain2
- If last two bits equal 11, place item in Domain3

This algorithm provides a distribution of items among domains, uniformity of which is directly controlled by the hash function. The additional advantage of this scheme is the ease with which it can be adjusted to partition your data among larger number of domains by considering more and more bits of the hash value (3 bits will distribute to 8 domains, 4 bits to 16 domains and so on).

AWS Identity and Access Management

is integrated with AWS Identity and Access Management (IAM), which offers a wide range of features:

- Create users and groups in your AWS account.
- Easily share your AWS resources between the users in your AWS account.
- Assign unique security credentials to each user.
- Control each user's access to services and resources.
- Get a single bill for all users in your AWS account.

For example, you can use IAM to control access to a specific domain that your AWS account owns.

For general information about IAM, go to:

- Identity and Access Management (IAM)
- AWS Identity and Access Management Getting Started Guide
- Using AWS Identity and Access Management

For specific information about how you can control User access to Amazon SimpleDB, see Managing Users of Amazon SimpleDB (p. 20).

Using Amazon SimpleDB

Topics

- Available Libraries (p. 13)
- Making API Requests (p. 13)
- Request Authentication (p. 17)
- Working with Domains (p. 29)
- Working with Data (p. 31)
- Conditionally Putting and Deleting Data (p. 33)
- Using Select to Create Amazon SimpleDB Queries (p. 36)
- Working with Numerical Data (p. 45)
- Tuning Queries (p. 47)
- Working with XML-Restricted Characters (p. 49)

This section describes major concepts you should understand before building your Amazon SimpleDB application.

Available Libraries

AWS provides libraries (the AWS SDKs), which include sample code, tutorials, and other resources for software developers who prefer to build applications using language-specific APIs instead of writing their own HTTP requests. These SDKs provide basic functions (not included in the APIs), such as request authentication, request retries, and error handling so that it is easier to get started. AWS SDKs are available for the following languages:

- Java
- PHP
- Python
- Ruby
- Windows and .NET

For links to the documentation for all AWS SDKs for supported languages, go to the Software Development Kits (SDKs) section on the AWS Documentation page.

For libraries and sample code in all languages, go to Sample Code & Libraries.

Making API Requests

Topics

- Region Endpoints (p. 14)
- Making REST Requests (p. 14)

This section describes how to make REST requests.

Region Endpoints

To improve latency and to store data in a location that meets your requirements, Amazon SimpleDB enables you to select different region endpoints.

For information about this Amazon SimpleDB regions and endpoints, go to Regions and Endpoints in the Amazon Web Services General Reference.

For example, to create a SimpleDB domain in Europe, you would generate a REST request similar to the following:

```
https://sdb.eu-west-1.amazonaws.com/?Action=CreateDomain
&DomainName=MyDomain
&<authentication parameters>
```

Each Amazon SimpleDB endpoint is entirely independent. For example, if you have two domains called "MyDomain," one in sdb.amazonaws.com and one in sdb.eu-west-1.amazonaws.com, they are completely independent and do not share any data.

Making REST Requests

Topics

- About REST Requests (p. 14)
- Structure of a GET Request (p. 14)
- Structure of a POST Request (p. 15)
- Using Parameters with REST (p. 15)
- Sample REST Requests (p. 15)

This section provides information on making REST requests with the Amazon SimpleDB web service.

About REST Requests

For Amazon SimpleDB requests, use HTTP GET requests that are URLs with query strings, or use HTTP POST requests with a body of query parameters. You can use either HTTPS or HTTP for your requests. If the length of the query string that you are constructing exceeds the maximum allowed length of an HTTP GET URL, use the HTTP POST method, instead.

The response is an XML document that conforms to a schema.

Structure of a GET Request

This guide presents the Amazon SimpleDB GET requests as URLs. The URL consists of:

- **Endpoint—**The Amazon SimpleDB endpoints, see Regions and Endpoints.
- **Action—**The action you want to perform. For example, creating a new domain (CreateDomain). For a complete list, see Operations (p. 54).
- **Parameters—**A set of parameters that might be specific to the operation, such as an `ItemName`, or common to all operations, such as your `AWSAccessKeyId`.
- **AWSAccessKeyId—** The Access Key ID associated with your account.
- **Version—**The current API version for Amazon SimpleDB.

- **Signature—**The signature authenticates your request to AWS and must be accompanied by a valid timestamp. For information about calculating the signature value and providing the correct timestamp, see HMAC-SHA Signature (p. 23).
- **SignatureVersion—**Currently for Amazon SimpleDB, this value should always be `2`.
- **SignatureMethod—**`HmacSHA256`.
- **Timestamp—**A valid time stamp (instead of an expiration time) within 15 minutes before or after the request, see About the Time Stamp (p. 29).

Structure of a POST Request

Amazon SimpleDB `POST` requests consists of:

- **HTTP Headers—**The following headers are required:

```
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: sdb.amazonaws.com
```

  The `Host` value is one of the Amazon SimpleDB endpoints, see Regions and Endpoints.
- **Action—**The action you want to perform. For example, creating a new domain (CreateDomain). For a complete list, see Operations (p. 54).
- **Parameters—**A set of parameters that might be specific to the operation, such as an `ItemName`.
- **AWSAccessKeyId—** The Access Key ID associated with your account.
- **Version—**The current API version for Amazon SimpleDB.
- **Signature—**The signature authenticates your request to AWS, see HMAC-SHA Signature (p. 23).
- **SignatureVersion—**Currently for Amazon SimpleDB, this value should always be `2`.
- **SignatureMethod—**`HmacSHA256`
- **Timestamp—**A valid time stamp (instead of an expiration time) within 15 minutes before or after the request, see About the Time Stamp (p. 29).

Using Parameters with REST

In a REST request, each parameter is separated with an ampersand (&). The following is an example of the DomainName parameter and ItemName parameter using the ampersand (&) separator.
`DomainName=MyDomain&ItemName=Item123`

Parameters that have specific properties start with the main parameter name (such as `Attribute`), a dot, a sequence number, a dot, and the property name (such as `Name`). For example: `&Attribute.1.Name=Color`.

> **Note**
> Format the parameters as defined by the HTML 4.01 specification (section 17.13.4) for `application/x-www-form-urlencoded`. Parameter names become control names, and their values become control values. The order is not significant. However, also note that Amazon SimpleDB is more strict than the specification about what is URL encoded.

Sample REST Requests

This section provides sample REST requests and responses.

REST Request as a URL

The following shows a REST request that puts three attributes and values for an item named Item123 into the domain named MyDomain.

Note
A valid request does not contain line breaks. The following request contains line breaks to show each parameter clearly.

```
https://sdb.amazonaws.com/?Action=PutAttributes
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&AWSAccessKeyId=your_access_key
&Version=2009-04-15
&Signature=valid_signature
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
```

REST Response

The following is the sample response:

```
<PutAttributesResponse>
  <ResponseMetadata>
    <StatusCode>Success</StatusCode>
    <RequestId>f6820318-9658-4a9d-89f8-b067c90904fc</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

REST Request using HTTP POST

The following shows a REST request that puts three attributes and values for an item named Item123 into the domain named MyDomain.

Note
A valid request does not contain line breaks in the body of the request. The following request contains line breaks to show each parameter clearly.

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: sdb.amazonaws.com

Action=PutAttributes
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&AWSAccessKeyId=your_access_key
&Version=2009-04-15
&Signature=valid_signature
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
```

REST Response

The following is the sample response:

```
<PutAttributesResponse>
```

```
  <ResponseMetadata>
    <StatusCode>Success</StatusCode>
    <RequestId>f6820318-9658-4a9d-89f8-b067c90904fc</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Request Authentication

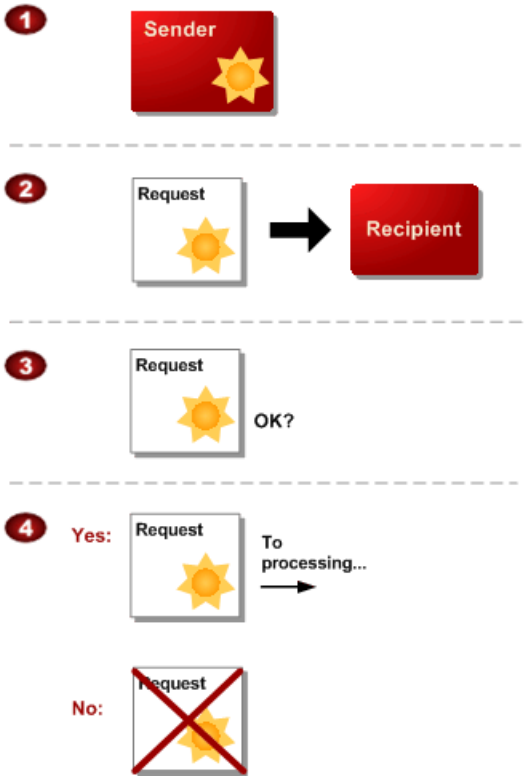
Topics

- What Is Authentication? (p. 18)
- Creating an AWS Account (p. 18)
- Managing Users of Amazon SimpleDB (p. 20)
- Using Temporary Security Credentials (p. 22)
- HMAC-SHA Signature (p. 23)

This section explains how Amazon SimpleDB authenticates your requests.

What Is Authentication?

Authentication is a process for identifying and verifying who is sending a request. The following diagram shows a simplified version of an authentication process.



General Process of Authentication

1	The sender obtains the necessary credential.
2	The sender sends a request with the credential to the recipient.
3	The recipient uses the credential to verify the sender truly sent the request.
4	If yes, the recipient processes the request. If no, the recipient rejects the request and responds accordingly.

During authentication, Amazon Web Services (AWS) verifies both the identity of the sender and whether the sender is registered to use services offered by AWS. If either test fails, the request is not processed further.

The subsequent sections describe how Amazon SimpleDB implements authentication to protect you and your customers' data.

Creating an AWS Account

To access any web service AWS offers, you must first create an AWS account at http://aws.amazon.com. You can use an existing Amazon.com account login and password when creating the AWS account.

From your AWS account you can view your AWS account activity, view usage reports, and manage your AWS Security Credentials.

To set up a new account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Managing Users of Amazon SimpleDB

Topics

- Amazon Resource Names (ARNs) for Amazon SimpleDB (p. 20)
- Amazon SimpleDB Actions (p. 21)
- Amazon SimpleDB Keys (p. 21)
- Example Policies for Amazon SimpleDB (p. 21)

Amazon SimpleDB does not offer its own resource-based permissions system. However, the service now integrates with IAM (AWS Identity and Access Management) so that you can give other Users in your AWS Account access to Amazon SimpleDB domains within the AWS Account. For example, Joe can create an Amazon SimpleDB domain, and then write an IAM policy specifying which Users in his AWS Account can access that domain. Joe can't give another AWS Account (or Users in another AWS Account) access to his AWS Account's SimpleDB domains.

> **Important**
> Aside from the integration with IAM, Amazon SimpleDB hasn't changed. Its API is not affected by the introduction of IAM, and includes no new actions related to Users and access control.

For examples of policies that cover Amazon SimpleDB actions and resources, see Example Policies for Amazon SimpleDB (p. 21).

Amazon Resource Names (ARNs) for Amazon SimpleDB

For Amazon SimpleDB, domains are the only resource type you can specify in a policy. The ARN format for domains follows this format:

```
arn:aws:sdb:<region>:<account_ID>:domain/<domain_name>
```

The *<region>* is required and can be any of the individual Regions Amazon SimpleDB supports (e.g., us-east-1), or * to represent all Regions. The *<region>* must not be blank.

Example

Following is an ARN for a domain named Domain1 in the us-east-1 region, belonging to AWS Account 111122223333.

```
arn:aws:sdb:us-east-1:111122223333:domain/Domain1
```

Example

Following is an ARN for a domain named Domain1 in all Regions that Amazon SimpleDB supports.

```
arn:aws:sdb:*:111122223333:domain/Domain1
```

You can use * and ? wildcards in the domain name. The * represents zero or multiple characters, and ? represents one character. For example, the following could refer to all the domains prefixed with `don_`.

```
arn:aws:sdb:*:111122223333:domain/don_*
```

For more information about ARNs, see ARNs.

Amazon SimpleDB Actions

In an IAM policy, you can specify any and all actions that Amazon SimpleDB offers. You must prefix each action name with the lowercase string `sdb:`. For example: `sdb:GetAttributes`, `sdb:Select`, `sdb:*` (for all Amazon SimpleDB actions). For a list of the actions, see Operations (p. 54).

Amazon SimpleDB Keys

Amazon SimpleDB implements the following policy keys, but no product-specific ones. For more information about policy keys, see Condition.

For a list of condition keys supported by each AWS service, see Actions, resources, and condition keys for AWS services in the *IAM User Guide*. For a list of condition keys that can be used in multiple AWS services, see AWS global condition context keys in the *IAM User Guide*.

Example Policies for Amazon SimpleDB

This section shows several simple policies for controlling User access to Amazon SimpleDB domains.

> **Note**
> In the future, Amazon SimpleDB might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

Example 1: Allow a group to use any Amazon SimpleDB actions on specific domains

In this example, we create a policy that lets the group use any of the AWS Account's domains that start with the literal string `test`.

```
{
   "Version": "2012-10-17",
   "Statement":[{
      "Effect":"Allow",
      "Action":"sdb:*",
      "Resource":"arn:aws:sdb:*:111122223333:domain/test*"
      }
   ]
}
```

Example 2: Allow a group to read data from the AWS Account's domains

In this example, we create a policy that lets the group use the `GetAttributes` and `Select` actions with any of the AWS Account's domains.

```
{
   "Version": "2012-10-17",
   "Statement":[{
      "Effect":"Allow",
      "Action":["sdb:GetAttributes","sdb:Select"],
      "Resource":"*"
      }
   ]
}
```

Example 3: Allow a group to list domains and get their metadata

In this example, we create a policy that lets the group use the `ListDomains` and `DomainMetadata` actions with any of the AWS Account's domains.

```
{
   "Version": "2012-10-17",
```

```
   "Statement":[{
      "Effect":"Allow",
      "Action":["sdb:ListDomains","sdb:DomainMetadata"],
      "Resource":"*"
      }
   ]
}
```

Example 4: Allow a partner to only read data from a particular domain

There's no way to share a domain with a different AWS Account, so the partner must work with your domain as a User within your own AWS Account.

In this example, we create an IAM User for the partner, and create a policy for the User that gives access to the `GetAttributes` and `Select` actions only on the domain named *mySDBDomain*.

(Instead of attaching the policy to the User, you could create a group for the partner, put the User in the group, and assign the policy to the group.)

You might also want to prevent the partner from doing anything else with mySDBDomain, so we add a statement that denies permission to any Amazon SimpleDB actions besides `GetAttributes` and `Select`. This is only necessary if there's also a broad policy that gives the AWS Account's Users wide access to Amazon SimpleDB and all the AWS Account's domains.

```
{
   "Version": "2012-10-17",
   "Statement":[{
         "Effect":"Allow",
         "Action":["sdb:GetAttributes","sdb:Select"],
         "Resource":"arn:aws:sdb:*:111122223333:domain/mySDBDomain"
      },
      {
         "Effect":"Deny",
         "Action":["sdb:GetAttributes","sdb:Select"],
         "Resource":"*"
      }
   ]
}
```

Using Temporary Security Credentials

In addition to creating IAM users with their own security credentials, IAM also enables you to grant temporary security credentials to any user to allow the user to access your AWS services and resources. You can manage users who have AWS accounts; these users are IAM users. You can also manage users for your system who do not have AWS accounts; these users are called federated users. Additionally, "users" can also be applications that you create to access your AWS resources.

You can use these temporary security credentials to make requests to Amazon SimpleDB. Replace your usual `AWSAccessKeyId` parameter with the one provided by IAM, add the IAM `SecurityToken` as a new parameter, and sign the request with the `SecretKeyId` provided by IAM. If you send requests using expired credentials Amazon SimpleDB denies the request.

For more information about IAM support for temporary security credentials, go to Granting Temporary Access to Your AWS Resources in *Using IAM*.

Example Using Temporary Security Credentials to Authenticate an Amazon SimpleDB Request

The following example demonstrates the wire protocol for using temporary security credentials to authenticate an Amazon SimpleDB request over HTTPS.

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
&DomainName=MyDomain
&ItemName=JumboFez
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=Signature calculated using the SecretKeyId provided by AWS Security Token
 Service
&SecurityToken=Security Token Value
```

Note
AWS provides support for temporary security credentials and session tokens in the AWS SDKs so you can implement temporary security credentials or session tokens with a specific programming language. Each SDK has its own instructions for implementing this feature. For a current list of AWS SDKs that support this feature, see Ways to Access the AWS Security Token Service. Non-AWS products and services should have their own documentation about supporting temporary credentials and session tokens, if available.

HMAC-SHA Signature

Topics

- Required Authentication Information (p. 23)
- Authentication Process (p. 24)
- Signing REST Requests (p. 27)
- About the Time Stamp (p. 29)

Required Authentication Information

When accessing Amazon SimpleDB using one of the AWS SDKs, the SDK handles the authentication process for you. For a list of available AWS SDKs supporting Amazon SimpleDB, see Available Libraries (p. 13).

However, when accessing Amazon SimpleDB using a REST request, you must provide the following items so the request can be authenticated.

Authentication

- **AWSAccessKeyId—**Your AWS account is identified by your Access Key ID, which AWS uses to look up your Secret Access Key.
- **Signature—**Each request must contain a valid HMAC-SHA signature, or the request is rejected.

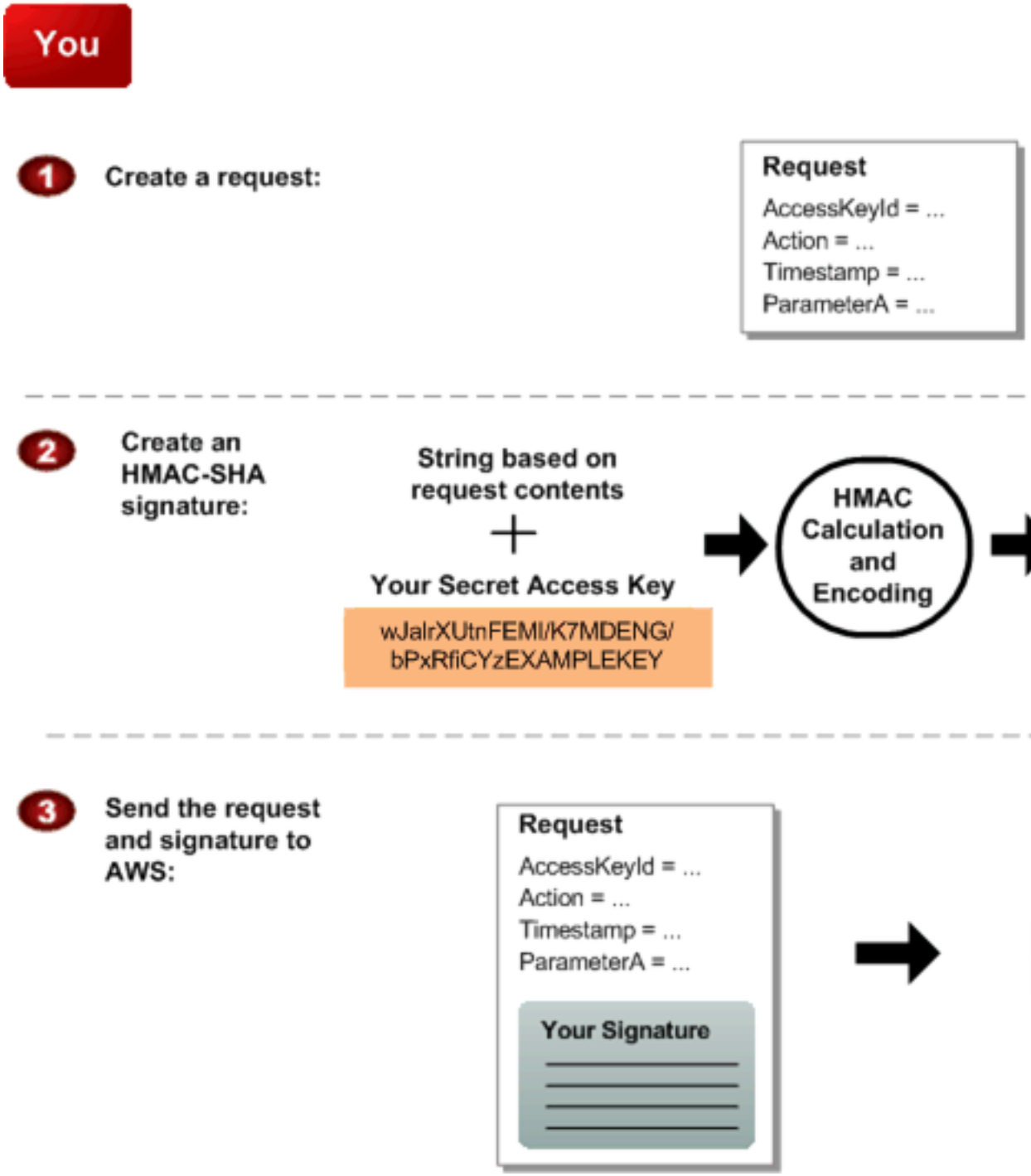
  A request signature is calculated using your Secret Access Key, which is a shared secret known only to you and AWS. You must use a HMAC-SHA256 signature.
- **Date—**Each request must contain the time stamp of the request.

  Depending on the API you're using, you can provide an expiration date and time for the request instead of or in addition to the time stamp. For details of what is required and allowed for each API, see the authentication topic for the particular API.

Authentication Process

Following is the series of tasks required to authenticate requests to AWS using an HMAC-SHA request signature. It is assumed you have already created an AWS account and received an Access Key ID and Secret Access Key. For more information about those, see Creating an AWS Account (p. 18).

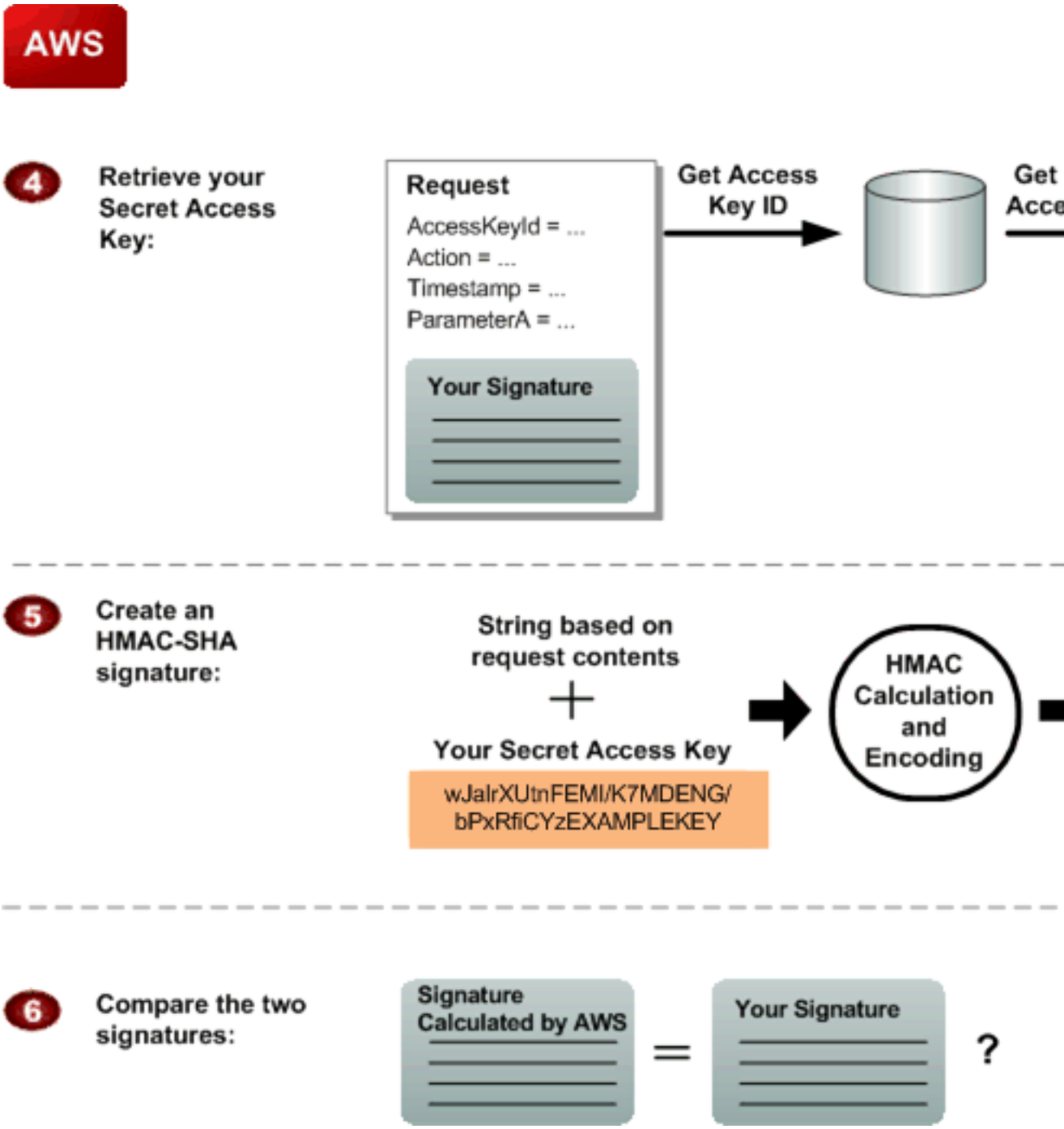
You perform the first three tasks.



Process for Authentication: Tasks You Perform

1	You construct a request to AWS.
2	You calculate a keyed-hash message authentication code (HMAC-SHA) signature using your Secret Access Key (for information about HMAC, go to http://www.rfc-editor.org/rfc/rfc2104.txt)
3	You include the signature and your Access Key ID in the request, and then send the request to AWS.

AWS performs the next three tasks.



Process for Authentication: Tasks AWS Performs

4	AWS uses the Access Key ID to look up your Secret Access Key.
5	AWS generates a signature from the request data and the Secret Access Key using the same algorithm you used to calculate the signature you sent in the request.

6	If the signature generated by AWS matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

Signing REST Requests

You can send REST requests over either HTTP or HTTPS. Regardless of which protocol you use, you must include a signature in every REST request. This section describes how to create the signature. The method described in the following procedure is known as *signature version 2*, and uses the HMAC-SHA256 signing method.

In addition to the requirements listed in Required Authentication Information (p. 23), signatures for REST requests must also include:

- **SignatureVersion—**The AWS signature version, which is currently the value `2`.
- **SignatureMethod—**Explicitly provide the signature method `HmacSHA256`.

Important
If you are currently using signature version 1: Version 1 is deprecated, and you should move to signature version 2 immediately.

To create the signature

1. Create the canonicalized query string that you need later in this procedure:
   a. Sort the UTF-8 query string components by parameter name with natural byte ordering.

      The parameters can come from the GET URI or from the POST body (when `Content-Type` is `application/x-www-form-urlencoded`).
   b. URL encode the parameter name and values according to the following rules:
      - Do not URL encode any of the unreserved characters that RFC 3986 defines. These unreserved characters are A-Z, a-z, 0-9, hyphen (-), underscore (_), period (.), and tilde (~).
      - Percent encode all other characters with %XY, where X and Y are hex characters 0-9 and uppercase A-F.
      - Percent encode extended UTF-8 characters in the form %XY%ZA....
      - Percent encode the space character as %20 (and not +, as common encoding schemes do).

      Note
      Currently all AWS service parameter names use unreserved characters, so you don't need to encode them. However, you might want to include code to handle parameter names that use reserved characters, for possible future use.
   c. Separate the encoded parameter names from their encoded values with the equals sign (=) (ASCII character 61), even if the parameter value is empty.
   d. Separate the name-value pairs with an ampersand (&) (ASCII character 38).
2. Create the string to sign according to the following pseudo-grammar (the `"\n"` represents an ASCII newline character).

```
StringToSign = HTTPVerb + "\n" +
               ValueOfHostHeaderInLowercase + "\n" +
               HTTPRequestURI + "\n" +
               CanonicalizedQueryString <from the preceding step>
```

The HTTPRequestURI component is the HTTP absolute path component of the URI up to, but not including, the query string. If the HTTPRequestURI is empty, use a forward slash (/).

3. Calculate an RFC 2104-compliant HMAC with the string you just created, your Secret Access Key as the key, and SHA256 or SHA1 as the hash algorithm.

   For more information, see http://www.ietf.org/rfc/rfc2104.txt.
4. Convert the resulting value to base64.
5. Use the resulting value as the value of the `Signature` request parameter.

   Important
   The final signature you send in the request must be URL encoded as specified in RFC 3986 (for more information, see http://www.ietf.org/rfc/rfc3986.txt). If your toolkit URL encodes your final request, then it handles the required URL encoding of the signature. If your toolkit doesn't URL encode the final request, then make sure to URL encode the signature before you include it in the request. Most importantly, make sure the signature is URL encoded *only once*. A common mistake is to URL encode it manually during signature formation, and then again when the toolkit URL encodes the entire request.

Some toolkits implement RFC 1738, which has different rules than RFC 3986 (for more information, go to http://www.rfc-editor.org/rfc/rfc2104.txt.

Example PutAttributes Request

```
https://sdb.amazonaws.com/?Action=PutAttributes
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&Version=2009-04-15
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
```

Following is the string to sign.

```
GET\n
sdb.amazonaws.com\n
/\n
AWSAccessKeyId=<Your AWS Access Key ID>
&Action=PutAttributes
&Attribute.1.Name=Color
&Attribute.1.Value=Blue
&Attribute.2.Name=Size
&Attribute.2.Value=Med
&Attribute.3.Name=Price
&Attribute.3.Value=0014.99
&DomainName=MyDomain
&ItemName=Item123
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&Version=2009-04-15
```

Following is the signed request.

```
https://sdb.amazonaws.com/?Action=PutAttributes
```

```
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&Version=2009-04-15
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&Signature=<URLEncode(Base64Encode(Signature))>
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
```

About the Time Stamp

The time stamp (or expiration time) you use in the request must be a `dateTime` object, with the complete date plus hours, minutes, and seconds (for more information, go to http://www.w3.org/TR/xmlschema-2/#dateTime). For example: 2010-01-31T23:59:59Z. Although it is not required, we recommend you provide the time stamp in the Coordinated Universal Time (Greenwich Mean Time) time zone.

If you specify a time stamp (instead of an expiration time), the request automatically expires 15 minutes after the time stamp (in other words, AWS does not process a request if the request time stamp is more than 15 minutes earlier than the current time on AWS servers). Make sure your server's time is set correctly.

> **Important**
> If you are using .NET you must not send overly specific time stamps, due to different interpretations of how extra time precision should be dropped. To avoid overly specific time stamps, manually construct `dateTime` objects with no more than millisecond precision.

Working with Domains

This section describes how to work create, list, and delete domains.

Topics

- Creating a Domain (p. 29)
- Verifying the Domain (p. 30)
- Deleting a Domain (p. 30)

Creating a Domain

The following is an example of creating a domain using REST.

```
https://sdb.amazonaws.com/
?Action=CreateDomain
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<CreateDomainResponse>
  <ResponseMetadata>
    <RequestId>2a1305a2-ed1c-43fc-b7c4-e6966b5e2727</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</CreateDomainResponse>
```

Verifying the Domain

The following is an example of listing domains using REST.

```
https://sdb.amazonaws.com/
?Action=ListDomains
&AWSAccessKeyId=[valid access key id]
&MaxNumberOfDomains=2
&NextToken=[valid next token]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A02%3A19-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<ListDomainsResponse>
  <ListDomainsResult>
    <DomainName>MyDomain</DomainName>
    <DomainName>MyOtherDomain</DomainName>
  </ListDomainsResult>
  <ResponseMetadata>
    <RequestId>eb13162f-1b95-4511-8b12-489b86acfd28</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</ListDomainsResponse>
```

Deleting a Domain

The following is an example of deleting a domain using REST.

```
https://sdb.amazonaws.com/
?Action=DeleteDomain
&AWSAccessKeyId=[valid access key id]
&DomainName=MyOtherDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A02%3A20-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<DeleteDomainResponse>
  <ResponseMetadata>
    <RequestId>c522638b-31a2-4d69-b376-8c5428744704</RequestId>
```

```
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</DeleteDomainResponse>
```

Working with Data

This section describes how to create, get, and delete attributes.

For detailed information about constructing queries, see Using Select to Create Amazon SimpleDB Queries (p. 36).

Topics

- Putting Data into a Domain (p. 31)
- Getting Data from a Domain (p. 32)
- Deleting Data from a Domain (p. 32)

Putting Data into a Domain

The following is an example of putting data into a domain using REST.

Note
When you put attributes, notice that the `Replace` parameter is optional, and set to `false` by default. If you do not explicitly set `Replace` to `true`, a new attribute name-value pair is created each time; even if the `Name` value already exists in your Amazon SimpleDB domain.

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&DomainName=MyDomain
&ItemName=JumboFez
&Attribute.1.Name=Color
&Attribute.1.Value=Blue
&Attribute.2.Name=Size
&Attribute.2.Value=Med
&Attribute.3.Name=Price
&Attribute.3.Value=0014.99
&Attribute.3.Replace=true
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Note
For information on performing multiple put operations at once, see BatchPutAttributes (p. 57).

Getting Data from a Domain

The following is an example of getting data from an item using REST.

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&ItemName=JumboFez
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<GetAttributesResponse>
  <GetAttributesResult>
    <Attribute><Name>Color</Name><Value>Blue</Value></Attribute>
    <Attribute><Name>Size</Name><Value>Med</Value></Attribute>
    <Attribute><Name>Price</Name><Value>0014.99</Value></Attribute>
  </GetAttributesResult>
  <ResponseMetadata>
    <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</GetAttributesResponse>
```

Deleting Data from a Domain

The following is an example of deleting data from an item using REST.

```
https://sdb.amazonaws.com/
?Action=DeleteAttributes
&DomainName=MyDomain
&ItemName=JumboFez
&Attribute.1.Name=color
&Attribute.1.Value=red
&Attribute.2.Name=color
&Attribute.2.Value=brick
&Attribute.3.Name=color
&Attribute.3.Value=garnet
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Amazon SimpleDB returns output similar to the following.

```
<DeleteAttributesResponse>
  <ResponseMetadata>
    <RequestId>05ae667c-cfac-41a8-ab37-a9c897c4c3ca</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
```

```
  </ResponseMetadata>
</DeleteAttributesResponse>
```

Note
For information on performing multiple delete operations at once, see BatchDeleteAttributes (p. 54).

Conditionally Putting and Deleting Data

This section describes how to update or delete data when a specific condition is met.

Topics

- Performing a Conditional Put (p. 33)
- Performing a Conditional Delete (p. 35)

Performing a Conditional Put

Conditional put enables you to insert or replace values for one or more attributes of an item if the existing value of an attribute matches a value that you specify. If the value does not match or is not present, the insert or update is rejected with a 409 (MultiValuedAttribute, ConditionalCheckFailed) or 404 (AttributeDoesNotExist) error code.

Conditional updates are useful for preventing lost updates when different sources concurrently write to the same item.

Note
Conditional puts can only match single-valued attributes.

Optimistic Concurrency Control

Applications can implement optimistic concurrency control (OCC) by maintaining a version number (or timestamp) attribute as part of an item and by performing a conditional update based on the value of this version number.

To set up optimistic concurrency control, configure each writer to specify the expected name and expected value in put requests. If the expected value changes between the time the writer reads and writes to that value, the writer does not perform the update. The writer can then read the update to the value and perform another write based on the change.

Note
All writers must use conditional updates or updates can be lost

In the following example, the application does a conditional update of the item's state and sets the value to "fuzzy" only if the value of VersionNumber is 30. If another application changes the value of VersionNumber between this read and write, so that it is no longer 30, the updates fails.

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&DomainName=MyDomain
&ItemName=JumboFez
&Attribute.1.Name=state
&Attribute.1.Value=fuzzy
&Attribute.1.Replace=true
&Attribute.2.Name=VersionNumber
&Attribute.2.Value=31
&Attribute.2.Replace=true
&Expected.1.Name=VersionNumber
```

```
&Expected.1.Value=30
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

If the condition is met, Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Counters

You can also use conditional puts to implement counters. For example, an application might issue a `GetAttributes` call to retrieve the current page counter for a web page and write the new value using `PutAttributes` only if no other host has updated the value.

To set up counters, configure each writer to specify the expected name and expected value in put requests. If the counter value changes between the time the writer reads and writes to that value, the writer does not update the counter. The writer can then read the counter value and perform another write based on the change.

> **Note**
> All writers must use conditional updates or updates can be lost
> When updating a counter, you can use eventually consistent or consistent reads. If a stale value is read, the write is rejected by the system.
> If a counter is updated frequently, make sure to re-read the updated counter value on failure.

In the following example, the application updates the counter if the value did not change between the read and the write. If another application changes the value of PageHits between this read and write, so that it is no longer 121, the updates fails.

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&DomainName=MyDomain
&ItemName=www.fezco.com
&Attribute.1.Name=PageHits
&Attribute.1.Value=122
&Attribute.1.Replace=true
&Expected.1.Name=PageHits
&Expected.1.Value=121
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

If the condition is met, Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
```

```
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Existence Check

You can use conditional puts to only put an attribute if it does not exist.

To perform an existence check, specify expected name and set expected exists to false. If the specified attribute does not exist, Amazon SimpleDB performs the update.

In the following example, Amazon SimpleDB creates the `quantity` attribute and sets its value to `144` for the `PetiteFez` item, if its `quantity` attribute does not exist.

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&DomainName=MyDomain
&ItemName=PetiteFez
&Attribute.1.Name=quantity
&Attribute.1.Value=144
&Expected.1.Name=quantity
&Expected.1.Exists=false
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

If the condition is met, Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Performing a Conditional Delete

Conditional delete enables you to delete an item or one or more attributes of an item if the existing value of an attribute matches a value that you specify. If value does not match or is not present, the insert or update is rejected with a 409 (MultiValuedAttribute, ConditionalCheckFailed) or 404 (AttributeDoesNotExist) error code.

Note
Conditional deletes can only match single-valued attributes.

To perform a conditional delete, specify the expected name and expected value for a `DeleteAttributes` operation. If the specified attribute does not exist, Amazon SimpleDB performs the delete.

In the following example, Amazon SimpleDB deletes the `JumboFez` product from the `MyDomain` domain if the quantity of the product reaches zero.

```
https://sdb.amazonaws.com/
```

```
?Action=DeleteAttributes
&DomainName=MyDomain
&ItemName=JumboFez
&Expected.1.Name=quantity
&Expected.1.Value=0
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

If the condition is met, Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Using Select to Create Amazon SimpleDB Queries

This section describes `Select`, a function that takes query expressions similar to the standard SQL `SELECT` statement.

Use the following format for the `Select` function.

```
select output_list
from domain_name
[where expression]
[sort_instructions]
[limit limit]
```

The *`output_list`* can be any of the following:

- * (all attributes)
- itemName() (the item name only)
- count(*)
- An explicit list of attributes (attribute1,..., attributeN)

Name	Description
`domain_name`	The domain to search.
`expression`	The match expression. This rest of this section provides examples of how to form select expressions.
`sort_instructions`	Sorts the results on a single attribute, in ascending or descending order. For information on sorting results, see Sort (p. 43).
`limit`	The *`limit`* is the maximum number of results per page to return (default: 100, max. 2500).

Name	Description
	Note The total size of the response cannot exceed 1 MB. Amazon SimpleDB automatically adjusts the number of items returned per page to enforce this limit. For example, even if you ask to retrieve 2500 items, but each individual item is 10 KB in size, the system returns 100 items and an appropriate next token so you can get the next page of results.

The *expression* can be any of the following:

- <select expression> intersection <select expression>
- NOT <select expression>
- (<select expression>)
- <select expression> or <select expression>
- <select expression> and <select expression>
- <simple comparison>

Note
For information on how to use quotes with Amazon SimpleDB, see Select Quoting Rules (p. 44).

Comparison Operators

Comparison operators are applied to a single attribute and are lexicographical in nature. When designing an application, you should carefully think through storing data in its appropriate string representation. For more information, see Working with Numerical Data (p. 45).

The following table shows all Amazon SimpleDB comparison operators.

Comparison Operator	Description	Select Example

=	Attribute value or itemName() equals the specified constant.	`select * from mydomain where city = 'Seattle'`
		
		`select * from mydomain where city = 'Seattle' or city = 'Portland'`

!=	Attribute value or itemName() does not equal to the specified constant.	`select * from mydomain where name != 'John'`
		
		`select * from mydomain where name != 'John' and name != 'Humberto'`

>	Attribute value or itemName() is greater than the specified constant.	`select * from mydomain where weight > '0034'`

Comparison Operator	Description	Select Example
>=	Attribute value or itemName() is greater than or equal to the specified constant.	`select * from mydomain where weight >= '065'`
<	Attribute value or itemName() is less than the specified constant.	`select * from mydomain where weight < '0034'`
<=	Attribute value or itemName() is less than or equal to the specified constant.	`select * from mydomain where year <= '2000'`
like	Attribute value or itemName() contains the specified constant. The `like` operator can be used to evaluate the start of a string ('*string*%'), the end of a string ('%*string*'), or any part of a string ('%*string*%'). **Note** Using the `like` operator to evaluate the end of a string or any part of a string is an expensive operation. Make sure to combine it with other predicates to reduce number of items it has to evaluate. To search for strings that contain the percent sign (%), you must escape it. For example, to search for a string that ends in '3%', you enter `select * from mydomain where name like '%3\%'`. To search for a string that begins with '3%', you enter `select * from mydomain where name like '3\%%'`. To search for a string that contains '3%', you enter `select * from mydomain where name like '%3\%%'`.	`select * from mydomain where author like 'Henry%' select * from mydomain where keyword = 'Book' and author like '%Miller'`
not like	Attribute value or itemName() does not contain the specified constant. The `not like` operator can be used to evaluate the start of a string ('*string*%'), the end of a string ('%*string*'), or any part of a string ('%*string*%'). **Note** Using the `not like` operator to evaluate the end of a string or any part of a string is an expensive operation. Make sure to combine it with other predicates to reduce number of items it has to evaluate.	`select * from mydomain where author not like 'Henry%' select * from mydomain where keyword = 'Book' and author not like '%Miller'`

Comparison Operator	Description	Select Example
between	Attribute value or itemName() falls within a range, including the start and end value.	`select * from mydomain where year between '1998' and '2000'`
in	Attribute value or itemName() is equal to one of the specified constants. When used with items, `in` acts as a batch get.	`select * from mydomain where year in('1998','2000','2003')` `select * from mydomain where itemName() in('0385333498','0802131786','B000T9886K')`
is null	Attribute does not exist. If an item has the attribute with an empty string, it is not returned. **Note** Due to performance issues, this operator is not recommended when most items have the specified attribute.	`select * from mydomain where year is null`
is not null	Attribute value or itemName() contains any value.	`select * from mydomain where year is not null`
every()	For multi-valued attributes, every attribute value must satisfy the constraint. **Note** Due to the cost of running more complex queries, this operator is only recommended for multi-valued attributes.	`select * from mydomain where every(keyword) = 'Book'` `select * from mydomain where every(keyword) like '***%'`

Sample Query Data Set

The following table contains the data set used throughout this section.

Item Name	Title	Author	Year	Pages	Keyword	Rating
0385333498	The Sirens of Titan	Kurt Vonnegut	1959	00336	Book Paperback	***** 5 stars Excellent
0802131786	Tropic of Cancer	Henry Miller	1934	00318	Book	****
1579124585	The Right Stuff	Tom Wolfe	1979	00304	Book Hardcover American	**** 4 stars
B000T9886K	In Between	Paul Van Dyk	2007		CD Trance	4 stars

Item Name	Title	Author	Year	Pages	Keyword	Rating
B00005JPLW	300	Zack Snyder	2007		DVD Action Frank Miller	*** 3 stars Not bad
B000SF3NGK	Heaven's Gonna Burn Your Eyes	Thievery Corporation	2002			*****

Simple Queries

This section shows simple queries and their results.

Note
To view the source data for the queries, see Sample Query Data Set (p. 39).

The following table shows some simple queries, how they are interpreted, and the results they return from the sample dataset.

Select Expression	Description	Result
`select * from mydomain where Title = 'The Right Stuff'`	Retrieves all items where the attribute "Title" equals "The Right Stuff".	1579124585
`select * from mydomain where Year > '1985'`	Retrieves all items where "Year" is greater than "1985". Although this looks like a numerical comparison, it is lexicographical. Because the calendar won't change to five digits for nearly 8,000 years, "Year" is not zero padded.	B000T9886K, B00005JPLW, B000SF3NGK
`select * from mydomain where Rating like '**** %'`	Retrieves all items that have at least a 4 star (****) rating. The prefix comparison is case-sensitive and exact and does not match attributes that only have the "4 star" value, such as item B000T9886K.	0385333498, 1579124585, 0802131786, B000SF3NGK
`select * from mydomain where Pages < '00320'`	Retrieves all items that have less than 320 pages. This attribute is zero padded in the data set and the select expression, which allows for proper lexicographical comparison between the strings. Items without this attribute are not considered.	1579124585, 0802131786,

Range Queries

Amazon SimpleDB enables you to execute more than one comparison against attribute values within the same predicate. This is most commonly used to specify a range of values.

This section shows range queries and their results.

Note
To view the source data for the queries, see Sample Query Data Set (p. 39).

The following table shows some range queries, how they are interpreted, and the results they return from the sample dataset.

Select Expression	Description	Result
`select * from mydomain where Year > '1975' and Year < '2008'`	Retrieves all items that have a "Year" value between "1975" and "2008", excluding "1975" and "2008".	1579124585, B000T9886K, B00005JPLW, B000SF3NGK
`select * from mydomain where Year between '1975' and '2008'`	Retrieves all items that have a "Year" value between "1975" and "2008", including "1975" and "2008".	1579124585, B000T9886K, B00005JPLW, B000SF3NGK
`select * from mydomain where Rating = '***' or Rating = '*****'`	Retrieves all items that have 3 (***) or 5 (*****) star rating This is a discontiguous range query that consists of two distinct values selected from the range of all possible values for the attribute.	0385333498, B00005JPLW, B000SF3NGK
`select * from mydomain where (Year > '1950' and Year < '1960') or Year like '193%' or Year = '2007'`	Retrieves all items where the "Year" attribute is either between "1950" and "1960", excluding "1950" and "1960", or falls in the nineteen-thirties, or equals "2007".	0385333498, 0802131786, B000T9886K, B00005JPLW

Queries on Attributes with Multiple Values

One of the unique features of Amazon SimpleDB is that it allows you to associate multiple values with a single attribute. Internet-related attributes such as *tag* or *keyword* often contain multiple values, which are easy to support through the Amazon SimpleDB data model and query language.

Important
Each attribute is considered individually against the comparison conditions defined in the predicate. Item names are selected if *any* of the values match the predicate condition. To change this behavior, use the `every()` operator to return results where *every* attribute matches the query expression.

This section shows queries on attributes with multiple values and their results.

Note
To view the source data for the queries, see Sample Query Data Set (p. 39).

The following table shows some queries on attributes with multiple values, how they are interpreted, and the results they return from the sample dataset.

Select Expression	Description	Result
`select * from mydomain where Rating = '4 stars' or Rating = '****'`	Retrieves all items with a 4 star (****) rating. The data set has this rating stored as both "4 stars" and "****." Amazon SimpleDB returns items that have either or both.	1579124585, 0802131786, B000T9886K

Select Expression	Description	Result
`select * from mydomain where Keyword = 'Book' and Keyword = 'Hardcover'`	Retrieve all items that have the Keyword attribute as both "Book" and "Hardcover." Based on the data set, you might be surprised that the result did not return the "1579124585" item. As described earlier, each value is evaluated individually against the predicate expression. Since neither of the values satisfies *both* comparisons defined in the predicate, the item name is not selected. To get the desired results, you can use the `select * from mydomain where Keyword = 'Book' intersection Keyword = 'Hardcover'` expression. For more information, see Multiple Attribute Queries (p. 42).	<none>
`select * from mydomain where every(keyword) in ('Book', 'Paperback')`	Retrieves all items where the only keyword is Book or Paperback. If the item contains any other keyword entries, it is not returned.	0385333498, 0802131786

Multiple Attribute Queries

The previous examples show how to create expressions for single predicates. The Amazon SimpleDB query language also supports constructing expressions across multiple predicates using the `intersection` operator.

Multiple attribute queries work by producing a set of item names from each predicate and applying the `intersection` operator. The intersection operator only returns item names that appear in both result sets.

This section shows multiple attribute queries and their results.

Note
To view the source data for the queries, see Sample Query Data Set (p. 39).

The following table shows some multiple attribute queries, how they are interpreted, and the results they return from the sample dataset.

Select Expression	Description	Result
`select * from mydomain where Rating = '****'`	Retrieves all items that have a "****" Rating.	0802131786, 1579124585
`select * from mydomain where every(Rating) = '****'`	Retrieves all items that only have a "****" Rating. Items are not returned that have a multi-valued Rating attribute that contains any value other than '****.'	0802131786
`select * from mydomain where Keyword = 'Book' intersection Keyword = 'Hardcover'`	Retrieves all items that have a "Book" Keyword and a "Hardcover" Keyword. The first predicate produces 0385333498, 0802131786, and 1579124585. The second	1579124585

Select Expression	Description	Result
	produces 1579124585. The intersection operator returns results that appear in both queries.	

Sort

Amazon SimpleDB supports sorting data on a single attribute or the item names, in ascending (default) or descending order. This section describes how to sort the result set returned from `Select`.

Note
All sort operations are performed in lexicographical order.
The sort attribute must be present in at least one of the predicates of the expression.
Because returned results must contain the attribute on which you are sorting, do not use `is null` on the sort attribute. For example, `select * from mydomain where author is null and title is not null order by title` will succeed. However, `select * from mydomain where author is null order by title` will fail because `title` is not constrained by the `not null` predicate.
To view the source data for the queries, see Sample Query Data Set (p. 39).

The following table shows sort queries, how they are interpreted, and the results they return from the sample dataset.

Select Expression	Description	Result
`select * from mydomain where Year < '1980' order by Year asc`	Retrieves all items released before 1980 and lists them in ascending order.	0802131786, 0385333498, 1579124585
`select * from mydomain where Year < '1980' order by Year`	Same as the previous entry, with "asc" (ascending) omitted.	0802131786, 0385333498, 1579124585
`select * from mydomain where Year = '2007' intersection Author is not null order by Author desc`	Retrieves all items released in 2007 and sorts them by author name in descending order.	B00005JPLW, B000T9886K
`select * from mydomain order by Year asc`	Invalid because Year is not constrained by a predicate in the where clause.	InvalidSortExpression error. See API Error Codes (p. 85).
`select * from mydomain where Year < '1980' order by Year limit 2`	Retrieves two items that were released before 1980 and lists them in ascending order.	0802131786, 0385333498
`select itemName() from mydomain where itemName() like 'B000%' order by itemName()`	Retrieves all itemNames() that start with B000 and lists them in ascending order.	B00005JPLW, B000SF3NGK, B000T9886K

Count

If you want to count the number of items in a result set instead of returning the items, use `count(*)`. Instead of returning a list of items, Amazon SimpleDB returns a single item called `Domain` with a `Count` attribute.

Note
If the count request takes more than five seconds, Amazon SimpleDB returns the number of items that it could count and a next token to return additional results. The client is responsible for accumulating the partial counts.
If Amazon SimpleDB returns a 408 Request Timeout, please resubmit the request.
The default result limit of 100 and maximum result limit of 2500 do not apply to count(*). However, you can restrict the maximum number of counted results using the limit clause.

The next token returned by `count(*)` and `select` are interchangeable as long as the `where` and `order by` clauses match. For example, if you want to return the 200 items after the first 10,000 (similar to an offset), you can perform a count with a limit clause of 10,000 and use the next token to return the next 200 items with `select`.

The following table shows `count(*)` queries and the results they return from the sample dataset.

Note
To view the source data for the queries, see Sample Query Data Set (p. 39).

Select Expression	Description	Result
`select count(*) from mydomain where Title = 'The Right Stuff'`	Counts all items where the attribute "Title" equals "The Right Stuff."	1
`select count(*) from mydomain where Year > '1985'`	Counts all items where "Year" is greater than "1985."	3
`select count(*) from mydomain limit 500`	Counts all items in the domain, with a limit of 500.	6
`select count(*) from mydomain limit 4`	Counts all items in the domain, with a limit of 4.	4

Select Quoting Rules

Attribute values must be quoted with a single or double quote. If a quote appears within the attribute value, it must be escaped with the same quote symbol. These following two expressions are equivalent:

```
select * from mydomain where attr1 = 'He said, "That''s the ticket!"'
select * from mydomain where attr1 = "He said, ""That's the ticket!"""
```

Attribute and domain names may appear without quotes if they contain only letters, numbers, underscores (_), or dollar symbols ($) and do not start with a number. You must quote all other attribute and domain names with the backtick (`).

```
select * from mydomain where `timestamp-1` > '1194393600'
```

You must escape the backtick when it appears in the attribute or domain name by replacing it with two backticks. For example, we can retrieve any items that have the attribute abc`123 set to the value 1 with this select expression:

```
select * from mydomain where `abc``123` = '1'
```

The following is the list of reserved keywords that are valid identifiers that must be backtick quoted if used as an attribute or domain name in the Select syntax.

- or
- and
- not
- from
- where
- select
- like
- null
- is
- order
- by
- asc
- desc
- in
- between
- intersection
- limit
- every

Working with Numerical Data

Topics

- Negative Numbers Offsets (p. 45)
- Zero Padding (p. 46)
- Dates (p. 46)

Amazon SimpleDB is a schema-less data store and everything is stored as a UTF-8 string value. This provides application designers with the flexibility of enforcing data restrictions at the application layer without the data store enforcing constraints.

All comparisons are performed lexicographically. As a result, we highly recommend that you use negative number offsets, zero padding, and store dates in an appropriate format.

Negative Numbers Offsets

When choosing a numerical range, ensure that every number is positive. To do this, choose an offset that is larger than the smallest expected negative number in your data set. For example, if the smallest expected number in your data set is -12,000, choosing offset = 100,000 might be safe.

The following is a sample original data set.

```
14.58, -12536.791, 20071109, 655378.34, -23
```

If you apply an offset of 100,000, the following is the resulting data set.

```
100014.58, 87463.209, 20171109, 755378.34, 99977
```

Zero Padding

After all the numbers in a data set are positive, ensure they are properly represented for lexicographical comparisons. For example, the string "10" comes before "2" in lexicographical order. If we zero pad the numbers to five digits, "00002" comes before "00010" and are compared correctly. Additionally, the offset is valid for numbers up to 5 digits and future numbers such as 00402 and 02987 are properly represented in this scheme.

To determine the right number of digits for zero padding, determine the largest number in your data set (accounting for negative number conversions), determine the offset number (maximum number of digits for that number without a decimal point), and convert all your numbers by appending zeros to them until they match the digit length of the offset number.

The following is sample data set with an offset applied.

```
100014.58, 87463.209, 20171109, 755378.34, 99977
```

If you zero pad the data set as well, the following is the resulting data set.

```
00100014.58, 00087463.209, 20171109, 00755378.34, 00099977
```

From this result set, the original query `'attribute' > '500'` is now `'attribute' > '00100500'`.

Dates

To convert dates to strings, we recommend following the ISO 8601 format, which supports lexicographical order comparisons.

The following table describes formats for representing date-time values with differing degrees of granularity. You must use components exactly as they are shown here and with exactly this punctuation. Note that the "T" appears literally in the string, to indicate the beginning of the time element, as is specified in ISO 8601.

Granularity	String
Year	YYYY (e.g., 1997)
Year and month	YYYY-MM (e.g., 1997-07)
Complete date	YYYY-MM-DD (e.g., 1997-07-16)
Complete date plus hours and minutes	YYYY-MM-DDThh:mmTZD (e.g., 1997-07-16T19:20+01:00)
Complete date plus hours, minutes and seconds	YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00)

Granularity	String
Complete date plus hours, minutes, seconds and a decimal fraction of a second	YYYY-MM-DDThh:mm:ss.sTZD (e.g., 1997-07-16T19:20:30.45+01:00)

Tuning Queries

Topics

- Tuning Your Queries Using Composite Attributes (p. 47)
- Data Set Partitioning (p. 48)

This section describes steps you can take to tune queries and your data set.

Tuning Your Queries Using Composite Attributes

Careful implementation of attributes can increase the efficiency of query operations in terms of duration and complexity. SimpleDB indexes attributes individually. In some cases, a query contains predicates on more than one attribute, and the combined selectivity of the predicates is significantly higher than the selectivity of each individual predicate. When this happens, the query retrieves a lot of data, and then removes most of the data to generate the result, which can degrade performance. If you find your queries using this pattern, you can implement composite attributes to improve your queries' performance.

The following example retrieves many books and many book prices before returning the requested result of books priced under nine dollars.

```
select * from myDomain where Type = 'Book' and Price < '9'
```

A composite attribute provides a more efficient way to handle this query. Assuming the attribute `Type` is a fixed four character string, a new composite attribute of `TypePrice` allows you to write a single predicate query.

```
select * from myDomain where TypePrice > 'Book' and TypePrice < 'Book9'
```

Performance for a multi-predicate query can also degrade if it uses an `order by` clause and the sorted attribute is constrained by a non-selective predicate. A typical example uses `not null`. For example, a table contains user names, billing timestamps, and a variety of other attributes. You want to get the latest 100 billing times for a user. A typical approach for this query leverages the index on the `user_id` attribute, retrieving all the records with the user's ID value, filtering the ones with correct values for the billing time, and then sorting the records and filtering out the top 100. The following example retrieves the latest 100 billing times for a user.

```
select * from myDomain where user_id = '1234' and bill_time is not null order by bill_time
 limit 100
```

However, if the predicate on `user_id` is not selective (i.e. many items exist in the domain for the `user_id` value `1234`), then the SimpleDB query processor could avoid dynamically sorting a very large number of records and scan the index on `bill_time`, instead. For this execution strategy, SimpleDB discards all the records not belonging to `user_id` value `1234`.

A composite attribute provides a more efficient way to handle this query, too. You can combine the `user_id` and `bill_time` values into a composite value, and then query for items with that value. The way you combine must depend on your data. In our example, bill_time may be a single string or may be missing, and the `user_id` attribute is a single four character string. We combine them by concatenating their texts; but if bill_time is missing, the missing data propagates and the concatenation is also missing. The following query would efficiently seek the billing times for a user by querying only that composite attribute.

```
select * from myDomain where user_id_bill_time like '1234%' order by user_id_bill_time
 limit 100
```

If `user_id` is a variable length field (not a fixed number of characters for the value), consider using a separator when combining it with `bill_time` in the `user_id_bill_time` composite attribute. For example, the following attribute assignment uses the vertical bar separator character (`|`) for a `user_id` that is six characters long: `user_id_bill_time = 123456|1305914378`. The following select example only gets the attributes with `user_id =1234` in the composite attribute, and does not get the attributes for the six character `user_id`.

```
select * from myDomain where user_id_bill_time like '1234|%' order by user_id_bill_time
 limit 100
```

The composite attribute technique is described further in the "Query performance optimization" section at Building for Performance and Reliability with Amazon SimpleDB.

Data Set Partitioning

Amazon SimpleDB allows up to 250 domains per subscriber. You can partition your data set among multiple domains to parallelize queries and operate on smaller data sets. Although you can only execute a single query against a single domain, you can aggregate result sets in the application layer.

> **Note**
> If you require additional domains, go to https://console.aws.amazon.com/support/home#/case/create?issueType=service-limit-increase&limitType=service-code-simpledb-domains.

You might choose to partition data sets across a natural dimension (e.g., product type, country). For example, you can keep a product catalog in a single domain, but it might be more efficient to partition it into "Book," "CD," and "DVD" domains. Additionally, you might need to partition data sets because your data requires higher throughput than a single domain, or the data set is very large and queries hit the timeout limit.

In some cases, data sets do not naturally present themselves well for partitioning (e.g., logs, events, or web-crawler data) and you might use a hashing algorithm to create a uniform distribution of items among multiple domains. For example, you could partition a data set into four different domains, determine the hash of items using a hash function such as MD5, and use the last two digits to place each item in the specified domain:

- Last two bits equal to 00: places item in Domain0
- Last two bits equal to 01: places item in Domain1
- Last two bits equal to 10: places item in Domain2
- Last two bits equal to 11: places item in Domain3

The additional advantage of this scheme is the ease with which it can be adjusted to partition your data across a larger number of domains by considering more and more bits of the hash value (3 bits distributes to 8 domains, 4 bits to 16 domains and so on).

Working with XML-Restricted Characters

You can store data in Amazon SimpleDB through the REST interface. All results are returned in XML documents.

XML does not support certain Unicode characters (the NUL character, anything in XML's RestrictedChar category, and permanently undefined Unicode characters). However, you can accidentally send them through the REST API. For more information about these characters, go to section 2.2 of the XML 1.1 specification.

To ensure that you can read all the data you sent via REST, if a response contains invalid XML characters, Amazon SimpleDB automatically Base64-encodes the UTF-8 octets of the text.

When a returned element is Base64-encoded, its encoding element is set to `base64`. The following example shows Base64-encoded results from a GetAttributes operation.

```
<GetAttributesResponse xmlns="http://sdb.amazonaws.com/doc/2009-04-15/">
  <GetAttributesResult>
    <Attribute>
      <Name>...</Name>
      <Value encoding="base64">...</Value>
    </Attribute>
    <Attribute>
      <Name encoding="base64">...</Name>
      <Value encoding="base64">...</Value>
    </Attribute>
  </GetAttributesResult>
 </GetAttributesResponse>
```

The following example shows a Base64-encoded result from a Select operation.

```
<SelectResponse xmlns="http://sdb.amazonaws.com/doc/2009-04-15/">
  <SelectResult>
    <Item>
      <Name>...</Name>
      <Attribute>
        <Name>...</Name>
        <Value encoding="base64">...</Value>
      </Attribute>
      <Attribute>
        <Name encoding="base64">...</Name>
        <Value encoding="base64">...</Value>
      </Attribute>
    </Item>
  </SelectResult>
 </SelectResponse>
```

When designing your application, make sure to scrub any data for invalid characters or design your application to handle Base64-encoded results.

API Reference

Topics

- API Usage (p. 50)
- Common Parameters (p. 52)
- Common Response Elements (p. 54)
- Common Error Responses (p. 54)
- Operations (p. 54)

This chapter contains detailed descriptions of all Amazon SimpleDB operations, their request parameters, their response elements, any special errors, and examples of requests and responses. All sample requests and responses are shown in a protocol-neutral format that displays the common elements for REST requests.

API Usage

This section provides a high-level overview of the Amazon SimpleDB API. It describes API conventions, API versioning used to minimize the impact of service changes, and API-specific information for making REST requests.

API Conventions

Overview

This topic discusses the conventions used in the Amazon SimpleDB API reference. This includes terminology, notation, and any abbreviations used to describe the API.

The API reference is broken down into a collection of *Actions* and *Data Types*.

Actions

Actions encapsulate the possible interactions with Amazon SimpleDB. These can be viewed as remote procedure calls and consist of a request and response message pair. Requests must be signed, allowing Amazon SimpleDB to authenticate the caller.

Data Types

Values provided as parameters to the various operations must be of the indicated type. Standard XSD types (like `string`, `boolean`, `int`) are prefixed with xsd:. Complex types defined by the Amazon SimpleDB WSDL are prefixed with `sdb:`.

WSDL Location and API Version

The Amazon SimpleDB API is published through a Web Services Description Language (WSDL) and an XML schema document. The version of the Amazon SimpleDB API supported with this document is 2009-04-15.

The Amazon SimpleDB WSDL is located at: http://sdb.amazonaws.com/doc/2009-04-15/AmazonSimpleDB.wsdl.

The Amazon SimpleDB schema is located at: http://sdb.amazonaws.com/doc/2009-04-15/AmazonSimpleDB.xsd.

Some libraries can generate code directly from the WSDL. Other libraries require a little more work on your part.

API Versions

All Amazon SimpleDB API operations are versioned. This minimizes the impact of API changes on client software by sending back a response that the client can process. New versions are designed to be backward-compatible with older API revisions. However, there might be occasions where an incompatible API change is required. Additionally, newer API responses might include additional fields and, depending on how the client software is written, it might not be able to handle additional fields. Including a version in the request guarantees that it will always be sent a response that it expects.

Each API revision is assigned a version in date form. This version is included in the request as a version parameter when using REST. The response returned by Amazon SimpleDB honors the version included in the request. Fields introduced in a later API version are not returned in the response.

The WSDL for each supported API version is available using the following URI format:

http://sdb.amazonaws.com/doc/<api-version>/AmazonSimpleDB.wsdl

Specifying the API Version

For all requests, you must explicitly request the API version you want to use. Specifying the version ensures that the service does not return response elements that your application is not designed to handle.

In REST requests, you include the Version parameter.

```
http://sdb.amazonaws.com
?Action=CreateDomain
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

API Error Retries

This section describes how to handle client and server errors.

Note
For information on specific error messages, see API Error Codes (p. 85)

Client Errors

REST client errors are indicated by a 4xx HTTP response code.

Do not retry client errors. Client errors indicate that Amazon SimpleDB found a problem with the client request and the application should address the issue before submitting the request again.

Server Errors

For server errors, you should retry the original request.

REST server errors are indicated by a 5xx HTTP response code.

Retries and Exponential Backoff

The AWS SDKs that support Amazon SimpleDB implement retries and exponential backoff. For more information, see Error Retries and Exponential Backoff in the AWS General Reference.

Common Parameters

This section describes parameters used by Amazon SimpleDB operations.

Some parameters are required by all operations and are not repeated in the documentation unless the usage is unique for that operation. Other parameters are conditional which indicates they are required for some operations and optional for others.

Request Parameters

Name	Description	Type
`Action`	Name of the action.	Required
`Attribute.X.Name` (REST)	Name of *attribute* associated with an item. Required by `PutAttributes` and `BatchPutAttributes`. Optional for `DeleteAttributes` and `BatchDeleteAttributes`.	Conditional
`Attribute.X.Value` (REST)	Value of attribute associated with an item. Required by `PutAttributes` and `BatchPutAttributes`. Optional for `DeleteAttributes` and `BatchDeleteAttributes`.	Conditional
`Attribute.X.Replace` (REST)	Flag to specify whether to replace the attribute/value or to add a new attribute/value. Used by `PutAttributes` and `BatchPutAttributes`. The default setting is `false`.	Optional
`AttributeName`	Name of attribute to return. Optional for `GetAttributes`.	Conditional
`AWSAccessKeyId`	For more information, see Request Authentication (p. 17).	Required
`DomainName`	Name of the domain used in the operation.	Required

Name	Description	Type
`ItemName`	Unique identifier of an item. Required by `PutAttributes`, `BatchPutAttributes`, `GetAttributes`, `DeleteAttributes`, and `BatchDeleteAttributes`.	Conditional
`MaxNumberOfDomains`	The maximum number of domain names you want returned. Used by `ListDomains`. The range is 1 to 100. The default setting is 100.	Optional
`MaxNumberOfItems`	Maximum number of items to return in the response. The range is 1 to 2500. The default setting is 100.	Optional
`NextToken`	String that tells Amazon SimpleDB where to start the next list of domain or item names. Used by `ListDomains` and `GetAttributes`.	Optional
`SelectExpression`	String that specifies the query that is executed against the domain.	Optional
`Signature`	For more information, see Signing REST Requests (p. 27).	Required
`SignatureMethod`	Required when you use signature version 2 with REST requests. For more information, see Signing REST Requests (p. 27).	Conditional
`SignatureVersion`	For more information, see Signing REST Requests (p. 27).	Required
`Timestamp`	For more information, see Request Authentication (p. 17).	Required
`Version`	Version of the API. The version of the API described in this document is 2009-04-15.	Required

Request Parameter Formats

The following are specifications for Amazon SimpleDB user data:

User Data Specifications

- **Domain names—**Allowed characters are a-z, A-Z, 0-9, '_', '-', and '.' .

  Domain names can be between 3 and 255 characters long.
- **Item names, attribute names, and attribute values—**Allowed characters are all UTF-8 characters valid in XML documents.

  Control characters and any sequences that are not valid in XML are returned Base64-encoded. For more information, see Working with XML-Restricted Characters (p. 49).

Quotes and escape characters

User data in query expressions must be enclosed in single quotes. If a single quote is used within the user data, it must be *escaped* using a backslash. If a backslash is used within user data, it must be escaped as well. Examples:

Original String	Escaped String
`'John's AWS account'`	`'John\'s AWS account'`
`c:\path\constant`	`'c:\\path\\constant'`

Common Response Elements

Name	Description
`RequestId`	A unique ID for tracking the request.
`BoxUsage`	The measure of machine utilization for this request. This does not include storage or transfer usage.

Common Error Responses

Request authentication errors are described in API Error Codes (p. 85). All other errors are listed with the appropriate operations.

Operations

Topics

- BatchDeleteAttributes (p. 54)
- BatchPutAttributes (p. 57)
- CreateDomain (p. 62)
- DeleteAttributes (p. 63)
- DeleteDomain (p. 68)
- DomainMetadata (p. 70)
- GetAttributes (p. 72)
- ListDomains (p. 75)
- PutAttributes (p. 76)
- Select (p. 81)

BatchDeleteAttributes

Description

Performs multiple `DeleteAttributes` operations in a single call, which reduces round trips and latencies. This enables Amazon SimpleDB to optimize requests, which generally yields better throughput.

Note
If you specify `BatchDeleteAttributes` without attributes or values, all the attributes for the item are deleted.
`BatchDeleteAttributes` is an idempotent operation; running it multiple times on the same item or attribute *doesn't* result in an error.
The `BatchDeleteAttributes` operation succeeds or fails in its entirety. There are no partial deletes.
You can execute multiple `BatchDeleteAttributes` operations and other operations in parallel. However, large numbers of concurrent `BatchDeleteAttributes` calls can result in Service Unavailable (503) responses.
This operation is vulnerable to exceeding the maximum URL size when making a REST request using the HTTP GET method.
This operation does not support conditions using `Expected.Name`, `Expected.Value`, or `Expected.Exists`.

The following limitations are enforced for this operation:

- 1 MB request size
- 25 item limit per `BatchDeleteAttributes` operation

Request Parameters

Name	Description	Required
`Item.Y.ItemName`	The name of the item. Type: String. Default: None.	Yes
`Item.Y.Attribute.X.Name`	The name of the attribute for the specified item. *Y* or *X* can be any positive integer or 0. If you specify `BatchDeleteAttributes` without attribute names or values, all the attributes for the item are deleted. Type: String. Default: None.	No
`Item.Y.Attribute.X.Value`	The value of the attribute for the specified item. *Y* or *X* can be any positive integer or 0. If an attribute value is specified, then the corresponding attribute name is required. Type: String. Default: None.	Yes
`DomainName`	The name of the domain in which to perform the operation. Type: String Default: None.	Yes

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
AttributeDoesNotExist	Attribute ("+ name + ") does not exist.
DuplicateItemName	Item item_name was specified more than once.
InvalidParameterValue	Value (" + value + ") for parameter Name is invalid.The empty string is an illegal attribute name.
InvalidParameterValue	Value (" + value + ") for parameter Name is invalid. Value exceeds maximum length of 1024.
InvalidParameterValue	Value (" + value + ") for parameter Item is invalid. Value exceeds max length of 1024.
InvalidParameterValue	Value (" + value + ") for parameter Value is invalid. Value exceeds maximum length of 1024.
InvalidWSDLVersion	Parameter (" + parameterName +") is only supported in WSDL version 2009-04-15 or beyond. Please upgrade to new version.
MissingParameter	The request must contain the parameter DomainName.
MissingParameter	The request must contain the parameter ItemName.
MissingParameter	The request must contain the attribute Name, if an attribute Value is specified.
NoSuchDomain	The specified domain does not exist.
NumberSubmittedItemsExceeded	Too many items in a single call. Up to 25 items per call allowed.
NumberSubmittedAttributesExceeded	Too many attributes for item itemName in a single call. Up to 256 attributes per call allowed.

Examples

Sample Request

In this example, the Jumbo Fez and Petite Fez have sold out in several colors. The following sample deletes the red, brick, and garnet values from the color attribute of the JumboFez item, and the pink and fuchsia values from the color attribute of the PetiteFez item.

```
https://sdb.amazonaws.com/
?Action=BatchDeleteAttributes
&Item.1.ItemName=JumboFez
&Item.1.Attribute.1.name=color&
&Item.1.Attribute.1.value=red&
```

```
&Item.1.Attribute.2.name=color&
&Item.1.Attribute.2.value=brick&
&Item.1.Attribute.3.name=color&
&Item.1.Attribute.3.value=garnet&
&Item.2.ItemName=PetiteFez
&Item.2.Attribute.1.name=color&
&Item.2.Attribute.1.value=pink&
&Item.2.Attribute.2.name=color&
&Item.2.Attribute.2.value=fuchsia&
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<BatchDeleteAttributesResponse">
  <ResponseMetadata>
    <RequestId>05ae667c-cfac-41a8-ab37-a9c897c4c3ca</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</BatchDeleteAttributesResponse>
```

Related Actions

- DeleteAttributes (p. 63)
- GetAttributes (p. 72)

BatchPutAttributes

Description

With the `BatchPutAttributes` operation, you can perform multiple `PutAttribute` operations in a single call. This helps you yield savings in round trips and latencies, and enables Amazon SimpleDB to optimize requests, which generally yields better throughput.

You can specify attributes and values for *items* using a combination of the `Item.Y.Attribute.X.Name` and `Item.Y.Attribute.X.Value` parameters. To specify attributes and values for the first item, you use `Item.1.Attribute.1.Name` and `Item.1.Attribute.1.Value` for the first attribute, `Item.1.Attribute.2.Name` and `Item.1.Attribute.2.Value` for the second attribute, and so on.

To specify attributes and values for the second item, you use `Item.2.Attribute.1.Name` and `Item.2.Attribute.1.Value` for the first attribute, `Item.2.Attribute.2.Name` and `Item.2.Attribute.2.Value` for the second attribute, and so on.

Amazon SimpleDB uniquely identifies attributes in an item by their name/value combinations. For example, a single item can have the attributes `{ "first_name", "first_value" }` and `{ "first_name", second_value" }`. However, it cannot have two attribute instances where both the `Item.Y.Attribute.X.Name` and `Item.Y.Attribute.X.Value` are the same.

Optionally, you can supply the `Replace` parameter for each individual attribute. Setting this value to `true` causes the new attribute value to replace the existing attribute value(s) if any exist. Otherwise,

Amazon SimpleDB simply inserts the attribute values. For example, if an item has the attributes `{ 'a', '1' }`, `{ 'b', '2'}`, and `{ 'b', '3' }` and the requester calls `BatchPutAttributes` using the attributes `{ 'b', '4' }` with the `Replace` parameter set to `true`, the final attributes of the item are changed to `{ 'a', '1' }` and `{ 'b', '4' }`. This occurs because the new `'b'` attribute replaces the old value.

Note
You cannot specify an empty string as an item or attribute name.
The `BatchPutAttributes` operation succeeds or fails in its entirety. There are no partial puts.
You can execute multiple `BatchPutAttributes` operations and other operations in parallel. However, large numbers of concurrent `BatchPutAttributes` calls can result in Service Unavailable (503) responses.
This operation is vulnerable to exceeding the maximum URL size when making a REST request using the HTTP GET method.
This operation does not support conditions using `Expected.Name`, `Expected.Value`, or `Expected.Exists`.

The following limitations are enforced for this operation:

- 256 attribute name-value pairs per item
- 1 MB request size
- 1 billion attributes per domain
- 10 GB of total user data storage per domain
- 25 item limit per `BatchPutAttributes` operation

Request Parameters

Name	Description	Required
`Item.Y.ItemName`	The name of the item. Type: String. Default: None.	Yes
`Item.Y.Attribute.X.Name`	The name of the attribute for the specified item. Y or X can be any positive integer or 0. Type: String. Default: None.	Yes
`Item.Y.Attribute.X.Value`	The value of the attribute for the specified item. Y or X can be any positive integer or 0. Type: String. Default: None.	Yes
`Item.Y.Attribute.X.Replace`	Flag to specify whether to replace the Attribute/Value or to add a new Attribute/Value. The `replace` parameter is more resource intensive than non-replace operations and is not recommended unless required. Y or X can be any positive integer or 0.	No

Name	Description	Required
	To reduce the request size and latencies, we recommend that you do not specify this request parameter at all. Type: Boolean. Default: `false`.	
`DomainName`	The name of the domain in which to perform the operation. Type: String Default: None.	Yes

Note
When using *eventually consistent* reads, a GetAttributes (p. 72) or Select (p. 81) request (read) immediately after a DeleteAttributes (p. 63) or PutAttributes (p. 76) request (write) might not return the updated data. Some items might be updated before others, despite the fact that the operation never partially succeeds. A *consistent read* always reflects all writes that received a successful response prior to the read. For more information, see Consistency (p. 7).

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
`DuplicateItemName`	Item `item_name` was specified more than once.
`InvalidParameterValue`	Value `value` for parameter Name is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value `value` for parameter Value is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value `value` for parameter Item is invalid. Value exceeds max length of 1024.
`InvalidParameterValue`	Value `value` for parameter Replace is invalid. The Replace flag should be either `true` or `false`.
`MissingParameter`	The request must contain the parameter `DomainName`.
`MissingParameter`	The request must contain the parameter `ItemName`.
`MissingParameter`	`Attribute.Value` missing for `Attribute.Name=attribute_name`.
`MissingParameter`	`Attribute.Name` missing for `Attribute.Value=attribute_name`.
`MissingParameter`	No attributes for item `item_name`.

Error	Description
NoSuchDomain	The specified domain does not exist.
NumberItemAttributesExceeded	Too many attributes in this item.
NumberDomainAttributesExceeded	Too many attributes in this domain.
NumberDomainBytesExceeded	Too many bytes in this domain.
NumberSubmittedItemsExceeded	Too many items in a single call. Up to 25 items per call allowed.
NumberSubmittedAttributesExceeded	Too many attributes for item itemName in a single call. Up to 256 attributes per call allowed.

Examples

Sample Request

The following example uses BatchPutAttributes on Shirt1, which has attributes (Color=Blue), (Size=Med), and (Price=0014.99) in MyDomain. If Shirt1 already had the Price attribute, this operation would replace the values for that attribute. Otherwise, a new (additional) Price attribute is created with the value 0014.99.

The example also uses BatchPutAttributes on Shirt2 which has attributes (Color=Red), (Size=Large), and (Price=0019.99).

```
https://sdb.amazonaws.com/
?Action=BatchPutAttributes
&Item.1.ItemName=Shirt1
&Item.1.Attribute.1.Name=Color
&Item.1.Attribute.1.Value=Blue
&Item.1.Attribute.2.Name=Size
&Item.1.Attribute.2.Value=Med
&Item.1.Attribute.3.Name=Price
&Item.1.Attribute.3.Value=0014.99
&Item.1.Attribute.3.Replace=true
&Item.2.ItemName=Shirt2
&Item.2.Attribute.1.Name=Color
&Item.2.Attribute.1.Value=Red
&Item.2.Attribute.2.Name=Size
&Item.2.Attribute.2.Value=Large
&Item.2.Attribute.3.Name=Price
&Item.2.Attribute.3.Value=0019.99
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2009-01-12T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<BatchPutAttributesResponse>
  <ResponseMetadata>
```

```
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</BatchPutAttributesResponse>
```

Related Actions

- PutAttributes (p. 76)
- DeleteAttributes (p. 63)
- GetAttributes (p. 72)

CreateDomain

Description

The `CreateDomain` operation creates a new domain. The domain name must be unique among the domains associated with the Access Key ID provided in the request. The `CreateDomain` operation might take 10 or more seconds to complete.

Note
`CreateDomain` is an idempotent operation; running it multiple times using the same domain name will *not* result in an error response.
You can create up to 250 domains per account.
If you require additional domains, go to https://console.aws.amazon.com/support/home#/case/create?issueType=service-limit-increase&limitType=service-code-simpledb-domains.

Request Parameters

Name	Description	Required
`DomainName`	The name of the domain to create. The name can range between 3 and 255 characters and can contain the following characters: a-z, A-Z, 0-9, '_', '-', and '.'. Type: String	Yes

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter DomainName is invalid.
`MissingParameter`	The request must contain the parameter `DomainName`.
`NumberDomainsExceeded`	Number of domains limit exceeded.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=CreateDomain
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<CreateDomainResponse>
  <ResponseMetadata>
    <RequestId>2a1305a2-ed1c-43fc-b7c4-e6966b5e2727</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</CreateDomainResponse>
```

Related Actions

- DeleteDomain (p. 68)
- ListDomains (p. 75)

DeleteAttributes

Description

Deletes one or more attributes associated with the item. If all attributes of an item are deleted, the item is deleted.

Note
If you specify `DeleteAttributes` without attributes or values, all the attributes for the item are deleted.
Unless you specify conditions, the `DeleteAttributes` is an idempotent operation; running it multiple times on the same item or attribute does *not* result in an error response.
Conditional deletes are useful for only deleting items and attributes if specific conditions are met. If the conditions are met, Amazon SimpleDB performs the delete. Otherwise, the data is not deleted.
When using *eventually consistent* reads, a GetAttributes (p. 72) or Select (p. 81) request (read) immediately after a DeleteAttributes (p. 63) or PutAttributes (p. 76) request (write) might not return the updated data. A *consistent read* always reflects all writes that received a successful response prior to the read. For more information, see Consistency (p. 7).
You can perform the expected conditional check on one attribute per operation.

Request Parameters

Name	Description	Required
`ItemName`	The name of the item. Type: String	Yes
`Attribute.X.Name`	The name of the *attribute*. X can be any positive integer or 0. If you specify `DeleteAttributes` without attribute names or values, all the attributes for the item are deleted. Type: String	No
`Attribute.X.Value`	The name of the attribute value (for *multi-valued attributes*). X can be any positive integer or 0. If an attribute value is	No

Name	Description	Required
	specified, then the corresponding attribute name is required. Type: String	
`DomainName`	The name of the domain in which to perform the operation. Type: String	Yes
`Expected.Name`	Name of the attribute to check. Type: String. Conditions: Must be used with the expected value or expected exists parameter. When used with the expected value parameter, you specify the value to check. When expected exists is set to `true` and it is used with the expected value parameter, it performs similarly to just using the expected value parameter. When expected exists is set to `false`, the operation is performed if the expected attribute is not present. Can only be used with single-valued attributes.	Conditional
`Expected.Value`	Value of the attribute to check. Type: String. Conditions: Must be used with the expected name parameter. Can be used with the expected exists parameter if that parameter is set to `true`. Can only be used with single-valued attributes.	Conditional
`Expected.Exists`	Flag to test the existence of an attribute while performing conditional updates. Type: Boolean. Conditions: Must be used with the expected name parameter. When set to `true`, this must be used with the expected value parameter. When set to `false`, this cannot be used with the expected value parameter. Can only be used with single-valued attributes.	Conditional

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
`AttributeDoesNotExist`	Attribute ("+ name + ") does not exist.
`ConditionalCheckFailed`	Conditional check failed. Attribute (" + name + ") value exists.
`ConditionalCheckFailed`	Conditional check failed. Attribute ("+ name +") value is ("+ value +") but was expected ("+ expValue +").
`ExistsAndExpectedValue`	`Expected.Exists=false` and `Expected.Value` cannot be specified together.
`IncompleteExpectedExpression`	If `Expected.Exists=true` or unspecified, then `Expected.Value` has to be specified.
`InvalidParameterValue`	Value (" + value + ") for parameter `Expected.Exists` is invalid. `Expected.Exists` should be either `true` or `false`.
`InvalidParameterValue`	Value (" + value + ") for parameter `Name` is invalid.The empty string is an illegal attribute name.
`InvalidParameterValue`	Value (" + value + ") for parameter `Name` is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter `Value` is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter `Item` is invalid. Value exceeds max length of 1024.
`InvalidWSDLVersion`	Parameter (" + parameterName +") is only supported in WSDL version 2009-04-15 or beyond. Please upgrade to new version.
`MissingParameter`	The request must contain the parameter `DomainName`.
`MissingParameter`	The request must contain the parameter `ItemName`.
`MissingParameter`	The request must contain the attribute `Name`, if an attribute `Value` is specified.
`MultipleExistsConditions`	Only one `Exists` condition can be specified.
`MultipleExpectedNames`	Only one `Expected.Name` can be specified.
`MultipleExpectedValues`	Only one `Expected.Value` can be specified.
`MultiValuedAttribute`	Attribute (" + name + ") is multi-valued. Conditional check can only be performed on a single-valued attribute.
`NoSuchDomain`	The specified domain does not exist.

Examples

Sample Request

In this example, the Jumbo Fez has sold out in several colors. The following deletes the `red`, `brick`, and `garnet` values from the `color` attribute of the `JumboFez` item.

```
https://sdb.amazonaws.com/
?Action=DeleteAttributes
&Attribute.1.Name=color
&Attribute.1.Value=red
&Attribute.2.Name=color
&Attribute.2.Value=brick
&Attribute.3.Name=color
&Attribute.3.Value=garnet
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&ItemName=JumboFez
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<DeleteAttributesResponse">
  <ResponseMetadata>
    <RequestId>05ae667c-cfac-41a8-ab37-a9c897c4c3ca</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</DeleteAttributesResponse>
```

Sample Request

In this example, the Micro Fez has sold out. The following deletes the Micro Fez if the quantity reaches 0

> **Note**
> For more examples of conditional operations, see Conditionally Putting and Deleting Data (p. 33).

```
https://sdb.amazonaws.com/
?Action=DeleteAttributes
&ItemName=MicroFez
&Expected.Name=quantity
&Expected.Value=0
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<DeleteAttributesResponse>
```

```
  <ResponseMetadata>
    <RequestId>05ae667c-cfac-41a8-ab37-a9c897c4c3ca</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</DeleteAttributesResponse>
```

Related Actions

- BatchDeleteAttributes (p. 54)
- GetAttributes (p. 72)
- PutAttributes (p. 76)

DeleteDomain

Description

The `DeleteDomain` operation deletes a domain. Any items (and their attributes) in the domain are deleted as well. The `DeleteDomain` operation might take 10 or more seconds to complete.

Note
Running `DeleteDomain` on a domain that does not exist or running the function multiple times using the same domain name will *not* result in an error response.

Request Parameters

Name	Description	Required
`DomainName`	The name of the domain to delete. Type: String	Yes

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
`MissingParameter`	The request must contain the parameter `DomainName`.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=DeleteDomain
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A02%3A20-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<DeleteDomainResponse>
  <ResponseMetadata>
    <RequestId>c522638b-31a2-4d69-b376-8c5428744704</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
```

```
</DeleteDomainResponse>
```

Related Actions

- CreateDomain (p. 62)
- ListDomains (p. 75)

DomainMetadata

Description

Returns information about the domain, including when the domain was created, the number of items and attributes, and the size of attribute names and values.

Request Parameters

Name	Description	Required
`DomainName`	The name of the domain for which to display metadata. Type: String	Yes

Response Elements

Name	Description
`Timestamp`	The data and time when metadata was calculated in Epoch (UNIX) time.
`ItemCount`	The number of all items in the domain.
`AttributeValueCount`	The number of all attribute name/value pairs in the domain.
`AttributeNameCount`	The number of unique attribute names in the domain.
`ItemNamesSizeBytes`	The total size of all item names in the domain, in bytes.
`AttributeValuesSizeBytes`	The total size of all attribute values, in bytes.
`AttributeNamesSizeBytes`	The total size of all unique attribute names, in bytes.

Special Errors

Error	Description
`MissingParameter`	The request must contain the parameter `DomainName`.
`NoSuchDomain`	The specified domain does not exist.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=DomainMetadata
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<DomainMetadataResponse>
  <DomainMetadataResult>
    <ItemCount>195078</ItemCount>
    <ItemNamesSizeBytes>2586634</ItemNamesSizeBytes>
    <AttributeNameCount >12</AttributeNameCount >
    <AttributeNamesSizeBytes>120</AttributeNamesSizeBytes>
    <AttributeValueCount>3690416</AttributeValueCount>
    <AttributeValuesSizeBytes>50149756</AttributeValuesSizeBytes>
    <Timestamp>1225486466</Timestamp>
  </DomainMetadataResult>
  <ResponseMetadata>
     <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
     <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</DomainMetadataResponse>
```

Related Actions

- CreateDomain (p. 62)
- ListDomains (p. 75)

GetAttributes

Description

Returns all of the attributes associated with the item. Optionally, the attributes returned can be limited to one or more specified attribute name parameters.

Amazon SimpleDB keeps multiple copies of each domain. When data is written or updated, all copies of the data are updated. However, it takes time for the update to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change. If eventually consistent reads are not acceptable for your application, use `ConsistentRead`. Although this operation might take longer than a standard read, it always returns the last updated value.

Note
If the item does not exist on the replica that was accessed for this operation, an empty set is returned.
If you specify `GetAttributes` without any attribute names, all the attributes for the item are returned.

Request Parameters

Name	Description	Required
`ItemName`	The name of the item.	Yes
`AttributeName`	The name of the attribute.	No
`DomainName`	The name of the domain in which to perform the operation. Type: String	Yes
`ConsistentRead`	When set to `true`, ensures that the most recent data is returned. For more information, see Consistency (p. 7) Type: Boolean Default: `false`	No

Response Elements

Name	Description
`<Attribute><Name>... </Name><Value>... </Value></Attribute>`	The name of the attribute and value.

Special Errors

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter Name is invalid. Value exceeds maximum length of 1024.

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter Item is invalid. Value exceeds max length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter ConsistentRead is invalid. The ConsistentRead flag should be either `true` or `false`.
`MissingParameter`	The request must contain the parameter `DomainName`.
`MissingParameter`	The request must contain the parameter `ItemName`.
`NoSuchDomain`	The specified domain does not exist.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&ItemName=Item123
&ConsistentRead=true
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<GetAttributesResponse>
  <GetAttributesResult>
    <Attribute><Name>Color</Name><Value>Blue</Value></Attribute>
    <Attribute><Name>Size</Name><Value>Med</Value></Attribute>
    <Attribute><Name>Price</Name><Value>14</Value></Attribute>
  </GetAttributesResult>
  <ResponseMetadata>
    <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</GetAttributesResponse>
```

Sample Request

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&ItemName=Item123
&AttributeName.0=Color
&AttributeName.1=Size
&SignatureVersion=2
&SignatureMethod=HmacSHA256
```

```
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<GetAttributesResponse>
  <GetAttributesResult>
    <Attribute><Name>Color</Name><Value>Blue</Value></Attribute>
    <Attribute><Name>Size</Name><Value>Med</Value></Attribute>
  </GetAttributesResult>
  <ResponseMetadata>
    <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</GetAttributesResponse>
```

Related Actions

- DeleteAttributes (p. 63)
- PutAttributes (p. 76)

ListDomains

Description

The `ListDomains` operation lists all domains associated with the Access Key ID. It returns domain names up to the limit set by `MaxNumberOfDomains`. A `NextToken` is returned if there are more than `MaxNumberOfDomains` domains. Calling `ListDomains` successive times with the `NextToken` returns up to `MaxNumberOfDomains` more domain names each time.

Request Parameters

Name	Description	Required
`MaxNumberOfDomains`	The maximum number of domain names you want returned. Type: String The range is 1 to 100. The default setting is 100.	No
`NextToken`	String that tells Amazon SimpleDB where to start the next list of domain names.	No

Response Elements

Name	Description
`DomainName`	Domain names that match the expression.
`NextToken`	An opaque token indicating that there are more than `MaxNumberOfDomains` domains still available.

Special Errors

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter `MaxNumberOfDomains` is invalid. `MaxNumberOfDomains` must be between 1 and 100.
`InvalidNextToken`	The specified next token is not valid.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=ListDomains
```

```
&AWSAccessKeyId=[valid access key id]
&MaxNumberOfDomains=2
&NextToken=[valid next token]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A02%3A19-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<ListDomainsResponse>
  <ListDomainsResult>
    <DomainName>Domain1-200706011651</DomainName>
    <DomainName>Domain2-200706011652</DomainName>
    <NextToken>TWV0ZXJpbmdUZXN0RG9tYWluMS0yMDA3MDYwMTE2NTY=</NextToken>
  </ListDomainsResult>
  <ResponseMetadata>
    <RequestId>eb13162f-1b95-4511-8b12-489b86acfd28</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</ListDomainsResponse>
```

Related Actions

- CreateDomain (p. 62)
- DeleteDomain (p. 68)

PutAttributes

Description

The `PutAttributes` operation creates or replaces attributes in an item. You specify new attributes using a combination of the `Attribute.X.Name` and `Attribute.X.Value` parameters. You specify the first *attribute* by the parameters `Attribute.1.Name` and `Attribute.1.Value`, the second attribute by the parameters `Attribute.2.Name` and `Attribute.2.Value`, and so on.

Attributes are uniquely identified in an item by their name/value combination. For example, a single item can have the attributes `{ "first_name", "first_value" }` and `{ "first_name", second_value" }`. However, it cannot have two attribute instances where both the `Attribute.X.Name` and `Attribute.X.Value` are the same.

Optionally, the requester can supply the `Replace` parameter for each individual attribute. Setting this value to `true` causes the new attribute value to replace the existing attribute value(s). For example, if an item has the attributes `{ 'a', '1' }`, `{ 'b', '2'}` and `{ 'b', '3' }` and the requester calls `PutAttributes` using the attributes `{ 'b', '4' }` with the `Replace` parameter set to `true`, the final attributes of the item are changed to `{ 'a', '1' }` and `{ 'b', '4' }`, which replaces the previous values of the `'b'` attribute with the new value.

Conditional updates are useful for ensuring multiple processes do not overwrite each other. To prevent this from occurring, you can specify the expected attribute name and value. If they match, Amazon SimpleDB performs the update. Otherwise, the update does not occur.

> **Note**
> Using `PutAttributes` to replace attribute values that do not exist will *not* result in an error response.

You cannot specify an empty string as an attribute name.
When using *eventually consistent* reads, a GetAttributes (p. 72) or Select (p. 81) request (read) immediately after a DeleteAttributes (p. 63) or PutAttributes (p. 76) request (write) might not return the updated data. A *consistent read* always reflects all writes that received a successful response prior to the read. For more information, see Consistency (p. 7).
You can perform the expected conditional check on one attribute per operation.

The following limitations are enforced for this operation:

- 256 total attribute name-value pairs per item
- One billion attributes per domain
- 10 GB of total user data storage per domain

Request Parameters

Name	Description	Required
`Attribute.X.Name`	The name of the attribute. X can be any positive integer or 0. Type: String.	Yes
`Attribute.X.Value`	The value of the attribute. X can be any positive integer or 0. Type: String.	Yes
`ItemName`	The name of the item. Type: String.	Yes
`Attribute.X.Replace`	Flag to specify whether to replace the Attribute/Value or to add a new Attribute/Value. X can be any positive integer or 0. Type: Boolean. Default: `false`.	No
`DomainName`	The name of the domain in which to perform the operation. Type: String	Yes
`Expected.Name`	Name of the attribute to check. Type: String. Conditions: Must be used with the expected value or expected exists parameter. When used with the expected value parameter, you specify the value to check. When expected exists is set to `true` and it is used with the expected value parameter, it performs similarly to just using the expected value parameter. When expected exists is set	Conditional

Name	Description	Required
	to `false`, the operation is performed if the expected attribute is not present. Can only be used with single-valued attributes.	
`Expected.Value`	Value of the attribute to check. Type: String. Conditions: Must be used with the expected name parameter. Can be used with the expected exists parameter if that parameter is set to `true`. Can only be used with single-valued attributes.	Conditional
`Expected.Exists`	Flag to test the existence of an attribute while performing conditional updates. Type: Boolean. Conditions: Must be used with the expected name parameter. When set to `true`, this must be used with the expected value parameter. When set to `false`, this cannot be used with the expected value parameter. Can only be used with single-valued attributes.	Conditional

Response Elements

See Common Response Elements (p. 54).

Special Errors

Error	Description
`AttributeDoesNotExist`	Attribute ("+ name + ") does not exist
`ConditionalCheckFailed`	Conditional check failed. Attribute (" + name + ") value exists.
`ConditionalCheckFailed`	Conditional check failed. Attribute ("+ name +") value is ("+ value +") but was expected ("+ expValue +")
`ExistsAndExpectedValue`	Expected.Exists=false and Expected.Value cannot be specified together
`IncompleteExpectedExpression`	If Expected.Exists=true or unspecified, then Expected.Value has to be specified
`InvalidParameterValue`	Value (" + value + ") for parameter Expected.Exists is invalid. Expected.Exists should be either `true` or `false`.

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter Name is invalid.The empty string is an illegal attribute name
`InvalidParameterValue`	Value (" + value + ") for parameter Value is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter Name is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter Value is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter Item is invalid. Value exceeds max length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter Replace is invalid. The Replace flag should be either `true` or `false`.
`InvalidWSDLVersion`	Parameter (" + parameterName +") is only supported in WSDL version 2009-04-15 or beyond. Please upgrade to new version
`MissingParameter`	The request must contain the parameter Name
`MissingParameter`	The request must contain the parameter `DomainName`.
`MissingParameter`	The request must contain the parameter `ItemName`.
`MissingParameter`	`Attribute.Value` missing for `Attribute.Name='<attribute name>'`.
`MissingParameter`	`Attribute.Name` missing for `Attribute.Value='<attribute value>'`.
`MultipleExistsConditions`	Only one Exists condition can be specified
`MultipleExpectedNames`	Only one Expected.Name can be specified
`MultipleExpectedValues`	Only one Expected.Value can be specified
`MultiValuedAttribute`	Attribute (" + name + ") is multi-valued. Conditional check can only be performed on a single-valued attribute
`NoSuchDomain`	The specified domain does not exist.
`NumberItemAttributesExceeded`	Too many attributes in this item.
`NumberDomainAttributesExceeded`	Too many attributes in this domain.
`NumberDomainBytesExceeded`	Too many bytes in this domain.

Examples

Sample Request

The following example uses `PutAttributes` on `Item123`, which has attributes `(Color=Blue)`, `(Size=Med)`, and `(Price=0014.99)` in `MyDomain`. If `Item123` already had the `Price` attribute, this operation would replace the values for that attribute.

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&Attribute.1.Name=Color
&Attribute.1.Value=Blue
&Attribute.2.Name=Size
&Attribute.2.Value=Med
&Attribute.3.Name=Price
&Attribute.3.Value=0014.99
&Attribute.3.Replace=true
&AWSAccessKeyId=[valid access key id]
&DomainName=MyDomain
&ItemName=Item123
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

Sample Request

The following example uses conditional updates to ensure that multiple processes do not overwrite each other's settings. For example, if two people are buying the `JumboFez` item at the same time, the following ensures that the inventory is decremented correctly.

Note
For more examples of conditional operations, see Conditionally Putting and Deleting Data (p. 33).

```
https://sdb.amazonaws.com/
?Action=PutAttributes
&DomainName=MyDomain
&ItemName=JumboFez
&Attribute.1.Name=quantity
&Attribute.1.Value=14
&Attribute.1.Replace=true
&Expected.Name=quantity
&Expected.Value=15
&AWSAccessKeyId=[valid access key id]
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A05-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

If the update condition is met, Amazon SimpleDB returns output similar to the following.

```
<PutAttributesResponse>
  <ResponseMetadata>
    <RequestId>490206ce-8292-456c-a00f-61b335eb202b</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

In this example, one of the servers updates the value and the other receives an error. The server that receives the error resubmits the request specifying a value of 13 and an expected value of 14, ensuring that the inventory is correctly set.

Related Actions

- DeleteAttributes (p. 63)
- GetAttributes (p. 72)

Select

Description

The `Select` operation returns a set of `Attributes` for `ItemNames` that match the select expression. `Select` is similar to the standard SQL `SELECT` statement.

Amazon SimpleDB keeps multiple copies of each domain. When data is written or updated, all copies of the data are updated. However, it takes time for the update to propagate to all storage locations. The data will eventually be consistent, but an immediate read might not show the change. If eventually consistent reads are not acceptable for your application, use `ConsistentRead`. Although this operation might take longer than a standard read, it always returns the last updated value.

The total size of the response cannot exceed 1 MB. Amazon SimpleDB automatically adjusts the number of items returned per page to enforce this limit. For example, even if you ask to retrieve 2500 items, but each individual item is 10 KB in size, the system returns 100 items and an appropriate next token so you can get the next page of results.

For information on how to construct select expressions, see Using Select to Create Amazon SimpleDB Queries (p. 36).

Note
Operations that run longer than 5 seconds return a time-out error response or a partial or empty result set. Partial and empty result sets contain a `NextToken` value, which allows you to continue the operation from where it left off.
Responses larger than one megabyte return a partial result set.
Your application should *not* excessively retry queries that return `QueryTimeout` errors. If you receive too many `QueryTimeout` errors, reduce the complexity of your query expression.
When designing your application, keep in mind that Amazon SimpleDB does not guarantee how attributes are ordered in the returned response.
For information about limits that affect Select, see Limits (p. 10).
The `select` operation is case-sensitive.

Request Parameters

Name	Description	Required
`SelectExpression`	The expression used to query the domain.	Yes

Name	Description	Required
	Type: String Default: None	
`ConsistentRead`	When set to `true`, ensures that the most recent data is returned. For more information, see Consistency (p. 7) Type: Boolean Default: `false`	No
`NextToken`	String that tells Amazon SimpleDB where to start the next list of ItemNames. Type: String Default: None	No

Response Elements

Name	Description
`<Item><Name>... </Name></Item>`	Item names that match the select expression.
`<Attribute> <Name>...</Name> <Value>...</Value> </Attribute>`	The name and value of the attribute.
`NextToken`	An opaque token indicating that more than `MaxNumberOfItems` matched, the response size exceeded 1 megabyte, or the execution time exceeded 5 seconds.

Special Errors

Error	Description
`InvalidParameterValue`	Value (" + value + ") for parameter `MaxNumberOfItems` is invalid. `MaxNumberOfItems` must be between 1 and 2500.
`InvalidParameterValue`	Value (" + value + ") for parameter AttributeName is invalid. Value exceeds maximum length of 1024.
`InvalidParameterValue`	Value (" + value + ") for parameter ConsistentRead is invalid. The ConsistentRead flag should be either `true` or `false`.
`InvalidNextToken`	The specified next token is not valid.
`InvalidNumberPredicates`	Too many predicates in the query expression.
`InvalidNumberValueTests`	Too many value tests per predicate in the query expression.
`InvalidQueryExpression`	The specified query expression syntax is not valid.

Error	Description
InvalidSortExpression	The sort attribute must be present in at least one of the predicates, and the predicate cannot contain the is null operator.
MissingParameter	The request must contain the parameter DomainName.
NoSuchDomain	The specified domain does not exist.
QueryTimeout	A timeout occurred when attempting to query domain <domain name> with query expression <query expression>. BoxUsage [<box usage value>]"
TooManyRequestedAttributes	Too many attributes requested.

Examples

Sample Request

```
https://sdb.amazonaws.com/
?Action=Select
&AWSAccessKeyId=[valid access key id]
&NextToken=[valid next token]
&SelectExpression=select%20Color%20from%20MyDomain%20where%20Color%20like%20%27Blue%25%27
&ConsistentRead=true
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A09-07%3A00
&Version=2009-04-15
&Signature=[valid signature]
```

Sample Response

```
<SelectResponse>
  <SelectResult>
    <Item>
      <Name>Item_03</Name>
      <Attribute><Name>Category</Name><Value>Clothes</Value></Attribute>
      <Attribute><Name>Subcategory</Name><Value>Pants</Value></Attribute>
      <Attribute><Name>Name</Name><Value>Sweatpants</Value></Attribute>
      <Attribute><Name>Color</Name><Value>Blue</Value></Attribute>
      <Attribute><Name>Color</Name><Value>Yellow</Value></Attribute>
      <Attribute><Name>Color</Name><Value>Pink</Value></Attribute>
      <Attribute><Name>Size</Name><Value>Large</Value></Attribute>
    </Item>
    <Item>
      <Name>Item_06</Name>
      <Attribute><Name>Category</Name><Value>Motorcycle Parts</Value></Attribute>
      <Attribute><Name>Subcategory</Name><Value>Bodywork</Value></Attribute>
      <Attribute><Name>Name</Name><Value>Fender Eliminator</Value></Attribute>
      <Attribute><Name>Color</Name><Value>Blue</Value></Attribute>
      <Attribute><Name>Make</Name><Value>Yamaha</Value></Attribute>
      <Attribute><Name>Model</Name><Value>R1</Value></Attribute>
    </Item>
  </SelectResult>
  <ResponseMetadata>
```

```
    <RequestId>b1e8f1f7-42e9-494c-ad09-2674e557526d</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</SelectResponse>
```

API Error Codes

Topics

- About Response Code 503 (p. 85)
- Amazon SimpleDB Error Codes (p. 85)

There are two types of error codes, client and server.

Client error codes are generally caused by the client and might be an authentication failure or an invalid domain; these errors are accompanied by a 4xx HTTP response code.

Server error codes are generally caused by a server-side issue and a large volume of server error codes should be reported to Amazon Web Services (including the request ID and the time when the request was issued); these errors are accompanied by a 5xx HTTP response code.

About Response Code 503

Typically, a large volume of server error codes (5xx) should be reported to Amazon Web Services with one exception: response code 503. A response code 503 indicates that applications are submitting too many requests to Amazon SimpleDB in a very brief span of time. So, while other server error codes (5xx) indicate a distinct server problem, a 503 response code does not indicate a problem with Amazon SimpleDB, specifically, and should be resolved on the client side.

To resolve response code 503, implement request retries in the client application with exponential backoff. For details, see API Error Retries (p. 51). Or, split your domain into multiple shards to achieve better parallelism and higher throughput.

Amazon SimpleDB Error Codes

The following table lists all Amazon SimpleDB error codes.

Error	Description	HTTP Status Code
`AccessFailure`	Access to the resource " + resourceName + " is denied.	403 Forbidden
`AttributeDoesNotExist`	Attribute ("+ name + ") does not exist	404 Not Found
`AuthFailure`	AWS was not able to validate the provided access keys.	403 Forbidden
`AuthMissingFailure`	AWS was not able to authenticate the request: access keys are missing.	403 Forbidden
`ConditionalCheckFailed`	Conditional check failed. Attribute (" + name + ") value exists.	409 Conflict
`ConditionalCheckFailed`	Conditional check failed. Attribute ("+ name +") value is ("+ value +") but was expected ("+ expValue +")	409 Conflict

Error	Description	HTTP Status Code
ExistsAndExpectedValue	Expected.Exists=false and Expected.Value cannot be specified together	400 Bad Request
FeatureDeprecated	The replace flag must be specified per attribute, not per item.	400 Bad Request
IncompleteExpectedExpression	If Expected.Exists=true or unspecified, then Expected.Value has to be specified	400 Bad Request
IncompleteSignature	The request signature does not conform to AWS standards.	400 Bad Request
InternalError	Request could not be executed due to an internal service error.	500 Internal Server Error
InvalidAction	The action " + actionName + " is not valid for this web service.	400 Bad Request
InvalidHTTPAuthHeader	The HTTP authorization header is bad, use " + correctFormat".	400 Bad Request
InvalidHttpRequest	The HTTP request is invalid. Reason: " + reason".	400 Bad Request
InvalidLiteral	Illegal literal in the filter expression.	400 Bad Request
InvalidNextToken	The specified next token is not valid.	400 Bad Request
InvalidNumberPredicates	Too many predicates in the query expression.	400 Bad Request
InvalidNumberValueTests	Too many value tests per predicate in the query expression.	400 Bad Request
InvalidParameterCombination	The parameter " + param1 + " cannot be used with the parameter " + param2".	400 Bad Request
InvalidParameterValue	Value (" + value + ") for parameter MaxNumberOfDomains is invalid. MaxNumberOfDomains must be between 1 and 100.	400 Bad Request
InvalidParameterValue	Value (" + value + ") for parameter MaxNumberOfItems is invalid. MaxNumberOfItems must be between 1 and 2500.	400 Bad Request
InvalidParameterValue	Value (" + value + ") for parameter MaxNumberOfDomains is invalid. MaxNumberOfDomains must be between 1 and 100.	400 Bad Request
InvalidParameterValue	Value (" + value + ") for parameter " + paramName + " is invalid. " + reason".	400 Bad Request

Error	Description	HTTP Status Code
`InvalidParameterValue`	Value (" + value + ") for parameter Name is invalid. Value exceeds maximum length of 1024.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter Value is invalid. Value exceeds maximum length of 1024.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter DomainName is invalid.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter Replace is invalid. The Replace flag should be either `true` or `false`.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter Expected.Exists is invalid. Expected.Exists should be either `true` or `false`.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter Name is invalid.The empty string is an illegal attribute name	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter Value is invalid. Value exceeds maximum length of 1024.	400 Bad Request
`InvalidParameterValue`	Value (" + value + ") for parameter ConsistentRead is invalid. The ConsistentRead flag should be either `true` or `false`.	400 Bad Request
`InvalidQueryExpression`	The specified query expression syntax is not valid.	400 Bad Request
`InvalidResponseGroups`	The following response groups are invalid: " + invalidRGStr.	400 Bad Request
`InvalidService`	The Web Service " + serviceName + " does not exist.	400 Bad Request
`InvalidSortExpression`	The sort attribute must be present in at least one of the predicates, and the predicate cannot contain the is null operator.	400 Bad Request
`InvalidURI`	The URI " + requestURI + " is not valid.	400 Bad Request
`InvalidWSAddressingProperty`	WS-Addressing parameter " + paramName + " has a wrong value: " + paramValue".	400 Bad Request
`InvalidWSDLVersion`	Parameter (" + parameterName +") is only supported in WSDL version 2009-04-15 or beyond. Please upgrade to new version	400 Bad Request

Error	Description	HTTP Status Code
MissingAction	No action was supplied with this request.	400 Bad Request
MissingParameter	The request must contain the specified missing parameter.	400 Bad Request
MissingParameter	The request must contain the parameter " + paramName".	400 Bad Request
MissingParameter	The request must contain the parameter ItemName.	400 Bad Request
MissingParameter	The request must contain the parameter DomainName.	400 Bad Request
MissingParameter	Attribute.Value missing for Attribute.Name='name'.	400 Bad Request
MissingParameter	Attribute.Name missing for Attribute.Value='value'.	400 Bad Request
MissingParameter	No attributes for item ='" + itemName + "'.	400 Bad Request
MissingParameter	The request must contain the parameter Name	400 Bad Request
MissingWSAddressingProperty	WS-Addressing is missing a required parameter (" + paramName + ")".	400 Bad Request
MultipleExistsConditions	Only one Exists condition can be specified	400 Bad Request
MultipleExpectedNames	Only one Expected.Name can be specified	400 Bad Request
MultipleExpectedValues	Only one Expected.Value can be specified	400 Bad Request
MultiValuedAttribute	Attribute (" + name + ") is multi-valued. Conditional check can only be performed on a single-valued attribute	409 Conflict
NoSuchDomain	The specified domain does not exist.	400 Bad Request
NoSuchVersion	The requested version (" + version + ") of service " + service + " does not exist.	400 Bad Request
NotYetImplemented	Feature " + feature + " is not yet available".	401 Unauthorized
NumberDomainsExceeded	The domain limit was exceeded.	409 Conflict
NumberDomainAttributes Exceeded	Too many attributes in this domain.	409 Conflict
NumberDomainBytesExceeded	Too many bytes in this domain.	409 Conflict

Error	Description	HTTP Status Code
`NumberItemAttributes Exceeded`	Too many attributes in this item.	409 Conflict
`NumberSubmitted AttributesExceeded`	Too many attributes in a single call.	409 Conflict
`NumberSubmittedAttributesExceeded`	Too many attributes for item `itemName` in a single call. Up to 256 attributes per call allowed.	409 Conflict
`NumberSubmittedItemsExceeded`	Too many items in a single call. Up to 25 items per call allowed.	409 Conflict
`RequestExpired`	Request has expired. " + paramType + " date is " + date".	400 Bad Request
`QueryTimeout`	A timeout occurred when attempting to query domain <domain name> with query expression <query expression>. BoxUsage [<box usage value>]".	408 Request Timeout
`ServiceUnavailable`	Service Amazon SimpleDB is busy handling other requests, likely due to too many simultaneous requests. Consider reducing the frequency of your requests, and try again. See About Response Code 503 (p. 85).	503 Service Unavailable
`TooManyRequestedAttributes`	Too many attributes requested.	400 Bad Request
`UnsupportedHttpVerb`	The requested HTTP verb is not supported: " + verb".	400 Bad Request
`UnsupportedNextToken`	The specified next token is no longer supported. Please resubmit your query.	400 Bad Request
`URITooLong`	The URI exceeded the maximum limit of "+ maxLength".	400 Bad Request

Amazon SimpleDB Glossary

account	AWS account associated with a particular developer.
attribute	Similar to columns on a spreadsheet, attributes represent categories of data that can be assigned to items.
consistent read	A consistent read (using Select or GetAttributes with ConsistentRead=true) returns a result that reflects all writes that received a successful response prior to the read.
domain	All Amazon SimpleDB information is stored in domains. Domains are similar to tables that contain similar data. You can execute queries against a domain, but cannot execute joins between domains. The name of the domain must be unique within the customer account.
eventually consistent read	An eventually consistent read (using Select or GetAttributes) might not reflect the results of a recently completed write (using PutAttributes, BatchPutAttributes, DeleteAttributes). Consistency is usually reached within a second; repeating a read after a short time should return the updated data.
exponential backoff	A strategy for reducing the load on the system and increasing the likelihood of repeated requests succeeding by incrementally decreasing the rate at which retries are executed. For example, client applications might wait up to 400 milliseconds before attempting the first retry, up to 1600 milliseconds before the second, up to 6400 milliseconds (6.4 seconds) before the third, and so on.
items	Similar to rows on a spreadsheet, items represent individual objects that contain one or more value-attribute pairs
item name	An identifier for an item. The identifier must be unique within the domain.
machine utilization	Charges based on the amount of machine capacity used to complete the particular request (SELECT, GET, PUT, etc.), normalized to the hourly capacity of a circa 2007 1.7 GHz Xeon processor. Machine Utilization is measured in Machine Hour increments.
multi-valued attribute	An attribute with more than one value.
network partition	A rare error condition where some Amazon SimpleDB computers cannot contact each other, but all other components are operating correctly. Normally this is repaired within seconds or minutes.
single-valued attribute	An attribute with one value.

value	Similar to cells on a spreadsheet, values represent instances of attributes for an item. An attribute might have multiple values.

Document History

The following table describes the documentation for this release of *Amazon SimpleDB*.

Relevant Dates to this History:

- **API version:** 2009-04-15
- **Lastest document update:** April 12, 2012

Change	Description	Date Changed
Removed incorrect note.	The Note in the Select operation description was incorrect and has been removed. For more information, see Select (p. 81).	April 12, 2012
Added handling response code 503 information.	Added instructions for handling server response code 503. For more information, see About Response Code 503 (p. 85).	February 20, 2012
Added HTTP POST request information	Added instructions for forming HTTP POST requests. For more information, see Making REST Requests (p. 14).	February 20, 2012
Revised AWS version 2 signing information	Revised AWS version 2 signing instructions. For more information, see HMAC-SHA Signature (p. 23).	February 20, 2012
Support for AWS Security Token Service	Amazon SimpleDB now supports the AWS Security Token Service. For more information, see Using Temporary Security Credentials (p. 22).	01 Sept 2011
SOAP support deprecated	Amazon SimpleDB no longer supports requests using SOAP.	01 Sept 2011
New Tuning Queries section in documentation	A section was added to the Tuning Queries topic covering the use of composite attributes to improve query performance. For more information see Tuning Your Queries Using Composite Attributes (p. 47).	20 May 2011
New link	This service's endpoint information is now located in the Amazon Web Services General Reference. For more information, go to Regions and Endpoints in the Amazon Web Services General Reference.	02 March 2011
BatchDeleteAttributes	Amazon SimpleDB can now perform multiple delete operations at once. For more information, see BatchDeleteAttributes (p. 54).	03 December 2010
Asia Pacific Region	Amazon SimpleDB now supports the Asia Pacific region. For more information, see Region Endpoints (p. 14).	28 April 2010
Consistent read	GetAttributes and Select can now perform consistent reads, which always return the most recently written data. For more information, see Consistency (p. 7).	24 February 2010

Change	Description	Date Changed
Conditional put	Amazon SimpleDB now supports conditional put, which enables you to perform a put if a specific condition is met. For more information, see Conditionally Putting and Deleting Data (p. 33) and PutAttributes (p. 76).	24 February 2010
Conditional delete	Amazon SimpleDB now supports conditional delete, which enables you to delete data if a specific condition is met. For more information, see Conditionally Putting and Deleting Data (p. 33) and DeleteAttributes (p. 63).	24 February 2010
New Data Center in Europe	Amazon SimpleDB is now available in Europe. For more information, see Region Endpoints (p. 14).	23 September 2009
contains	You can now search whether attribute values contain a specified string using `like`. For more information, see Comparison Operators (p. 37).	18 May 2009
Sort and Execute Queries by itemName()	`Select` can now use `where` and `order by` with itemNames(). For more information, see Comparison Operators (p. 37).	18 May 2009
IS NULL Sort	Sort can now be applied to expressions that contain the `is null` predicate operator, as long as `is null` is not applied to the attribute that being sorted on. For more information about `Select`, see Sort (p. 43).	18 May 2009
Increased Item Limit	`Select` can now return up to 2500 items. For more information about `Select`, see Limits (p. 10).	18 May 2009
Query and QueryWithAttributes Deprecated	Amazon SimpleDB replaced `Query` and `QueryWithAttributes` with `Select`, a query function that is similar to the standard SQL `SELECT` statement. For more information, see Using Select to Create Amazon SimpleDB Queries (p. 36).	18 May 2009
SSL Required	All requests to Amazon SimpleDB must be made over SSL (https://). For more information, see Request Authentication (p. 17).	18 May 2009