AWS SDK for .NET Developer Guide

May 29, 2017

Contents

AWS SDK for .NET Developer Guide v3.0.0

This documentation is for version 3.0 of the AWS SDK for .NET, which is the latest version.

AWS SDK for .NET Developer Guide

The AWS SDK for .NET makes it easier for Windows developers to build .NET applications that tap into the cost-effective, scalable, and reliable AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2). The SDK supports development on any platform that supports the .NET Framework 3.5 or later, and you can develop applications with the SDK using Visual Studio 2010 or later.

What's in the SDK

The AWS SDK for .NET includes the following:

- The current version of the AWS SDK for .NET
- All previous major versions of the AWS SDK for .NET
- Sample code that demonstrates how to use the AWS SDK for .NET with several AWS services

To simplify installation, AWS provides the AWS Tools for Windows, which is a Windows installation package that includes:

- The AWS SDK for .NET
- The AWS Tools for Windows PowerShell (see the *Tools for Windows PowerShell User Guide*)
- The AWS Toolkit for Visual Studio (see the *Toolkit for Visual Studio User Guide*)

As an alternative to installing the AWS Tools for Windows, you can use NuGet to download individual AWSSDK service assemblies for a specific application project. For more information, see Installing AWSSDK Assemblies with NuGet.

Note

We recommend using Visual Studio Professional 2010 or later to implement your applications.

How to Use This Guide

The *AWS SDK for .NET Developer Guide* describes how to implement applications for AWS using the AWS SDK for .NET, and includes the following:

Getting Started with the AWS SDK for .NET

How to install and configure the AWS SDK for .NET. If you have not used the AWS SDK for .NET before or are having trouble with its configuration, you should start here.

Programming with the AWS SDK for .NET

The basics of how to implement applications with the AWS SDK for .NET that applies to all AWS services. This section also includes information about how to migrate code to the latest version of the AWS SDK for .NET, and describes the differences between the last version and this one.

AWS SDK for .NET Examples

A set of tutorials, walkthroughs, and examples showing how to use the AWS SDK for .NET to create applications for particular AWS services.

Additional Resources

More resources outside of this guide that provide valuable information about AWS and the AWS SDK for .NET.

A related document, *AWS SDK for .NET API Reference*, provides a detailed description of each namespace and class.

Supported Services and Revision History

The AWS SDK for .NET supports most AWS infrastructure products, and more services are added frequently. For a list of the AWS services supported by the SDK, see the SDK README file.

To see what changed in a given release, see the SDK change log.

Getting Started with the AWS SDK for .NET

To get started with the AWS SDK for .NET, complete the following tasks:

Create an AWS Account and Credentials

To use the AWS SDK for .NET to access AWS, you need an AWS account and AWS credentials. To increase the security of your AWS account, we recommend that you use an *IAM user* to provide access credentials instead of using your root account credentials.

Tip

For an overview of IAM users and why they are important for the security of your account, see Overview of Identity Management: Users in the *IAM User Guide*.

Signing Up for an AWS Account

To sign up for an AWS account

1. Open http://aws.amazon.com/ and click **Sign Up**.
2. Follow the on-screen instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone keypad.

Next, create an IAM user and download (or copy) its secret access key. To use the AWS SDK for .NET, you must have a set of valid AWS credentials, which consist of an access key and a secret key. These keys are used to sign programmatic web service requests and enable AWS to verify that the request comes from an authorized source. You can obtain a set of account credentials when you create your account. However, we recommend that you do not use these credentials with AWS SDK for .NET. Instead, create one or more IAM users, and use those credentials. For applications that run on Amazon EC2 instances, you can use IAM roles to provide temporary credentials.

Creating an IAM User

To create an IAM user

1. Go to the IAM console (you may need to sign in to AWS first).
2. Click **Users** in the sidebar to view your IAM users.
3. If you don't have any IAM users set up, click **Create New Users** to create one.
4. Select the IAM user in the list that you'll use to access AWS.
5. Open the **Security Credentials** tab, and click **Create Access Key**.

Note

You can have a maximum of two active access keys for any given IAM user. If your IAM user has two access keys already, then you'll need to delete one of them before creating a new key.

6. In the resulting dialog box, choose **Download Credentials** to download the credential file to your computer, or click **Show User Security Credentials** to view the IAM user's access key ID and secret access key (which you can copy and paste).

Important

There is no way to obtain the secret access key once you close the dialog. You can, however, delete its associated access key ID and create a new one.

Next, you set your credentials in the AWS shared credentials file or in the environment.

The preferred approach for handling credentials is to create a profile for each set of credentials in the SDK Store. You can create and manage profiles with the AWS Toolkit for Visual Studio, PowerShell cmdlets, or programmatically with the AWS SDK for .NET. These credentials are encrypted and stored separately from any project. You then reference the profile by name in your application, and the credentials are inserted at build time. This approach ensures that your credentials are not unintentionally exposed with your project on a public site. For more information, see Setting Up the AWS Toolkit for Visual Studio and Configuring AWS Credentials.

For more information about managing your credentials, see Best Practices for Managing AWS Access Keys.

To view your current account activity and manage your account at any time, go to http://aws.amazon.com and choose **My Account/Console**.

Install the .NET Development Environment

To use the AWS SDK for .NET, you must have the following installed.

Required

- Microsoft .NET Framework 3.5 or later
- Microsoft Visual Studio 2010 or later

Note

We recommend using Visual Studio Professional 2010 or later to implement your applications.

- AWS SDK for .NET

Note

The AWS SDK for .NET is installed with the AWS Toolkit for Visual Studio, a plugin that provides a user interface for managing your AWS resources from Visual Studio, and also includes the AWS Tools for Windows PowerShell.

For more information, see Using the AWS Toolkit for Visual Studio.

Install AWSSDK Assemblies

You can install the AWSSDK assemblies by installing the AWS SDK for .NET or by installing the AWS assemblies with NuGet.

Installing the AWS SDK for .NET

The following procedure describes how to install the AWS Tools for Windows, which contains the AWS SDK for .NET and the Tools for Windows PowerShell.

Tip

The AWS SDK for .NET is also available on GitHub.

To install AWS Tools for Windows

1. Go to AWS SDK for .NET.
2. In the **Downloads** section, choose **AWS SDK for .NET** to download the installer.
3. To start installation, run the downloaded installer and follow the on-screen instructions.

   ### Tip

   By default, AWS Tools for Windows are installed in the Program Files directory, which requires administrator privileges. To install AWS Tools for Windows as a non-administrator, choose a different installation directory.

4. (Optional) You can use NuGet to install individual AWSSDK service assemblies and extensions for the AWS SDK for .NET, which include a session state provider and a trace listener. For more information, see Installing AWSSDK Assemblies with NuGet.

Installing AWSSDK Assemblies with NuGet

NuGet is a package management system for the .NET platform. With NuGet, you can add the AWSSDK assemblies, as well as the TraceListener and SessionProvider extensions, to your application.

NuGet always has the most recent versions of the AWSSDK assemblies, and also enables you to install previous versions. NuGet is aware of dependencies between assemblies and installs all required assemblies automatically. Assemblies installed with NuGet are stored with your solution instead of in a central location, such as in the Program Files directory. This enables you to install assembly versions specific to a given application without creating compatibility issues for other applications. For more information about NuGet, see the NuGet documentation.

NuGet is installed automatically with Visual Studio 2010 or later. If you are using an earlier version of Visual Studio, you can install NuGet from the Visual Studio Gallery on MSDN.

You can use NuGet from **Solution Explorer** or from the Package Manager Console.

NuGet AWSSDK Packages

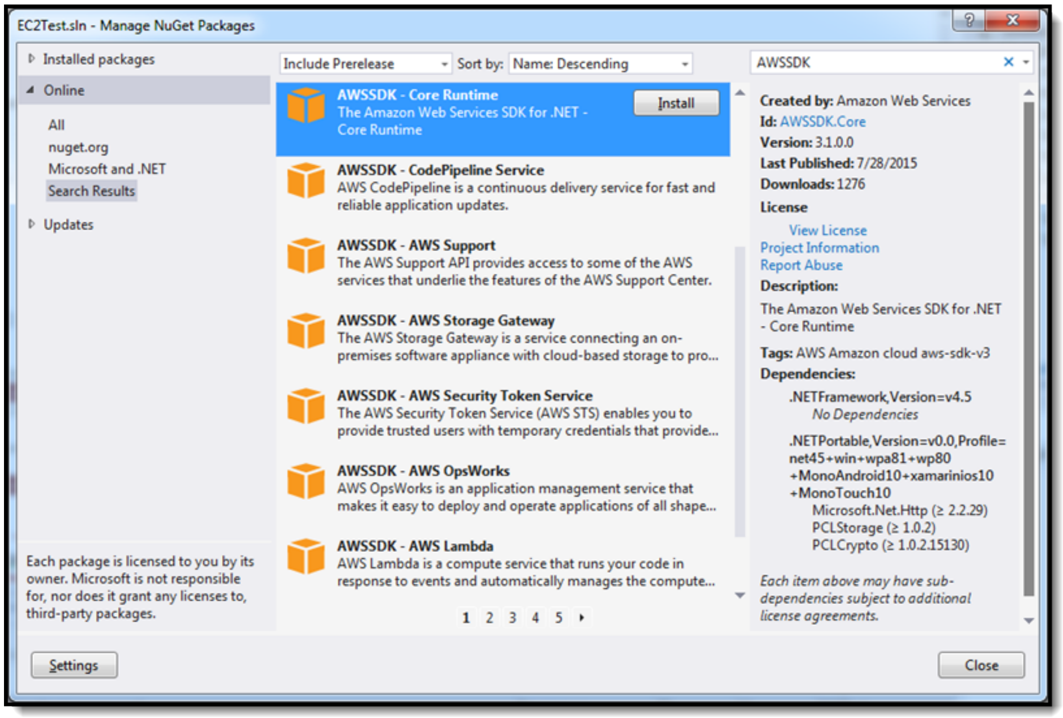
The NuGet website provides a page for every package available through NuGet. The page for each package includes a sample command line for installing the package using the Package Manager Console. Each page also includes a list of the previous versions of the package that are available through NuGet. For a list of AWSSDK packages available from NuGet, see AWSSDK Packages.

Using NuGet from Solution Explorer

To use NuGet from Solution Explorer

1. In **Solution Explorer**, right-click your project, and then choose **Manage NuGet Packages** from the context menu.
2. In the left pane of the **Manage NuGet Packages** dialog box, choose **Online**. You can then use the search box in the top-right corner to search for the package you want to install.

   The following figure shows the **AWSSDK - Core Runtime** assembly package. You can see NuGet is aware that this package has a dependency on the `AWSSDK.Core` assembly package; NuGet automatically installs the `AWSSDK.Core` package, if it is not already installed.



AWSSDK - Core Runtime
The Amazon Web Services SDK for .NET - Core Runtime

AWSSDK - CodePipeline Service
AWS CodePipeline is a continuous delivery service for fast and reliable application updates.

AWSSDK - AWS Support
The AWS Support API provides access to some of the AWS services that underlie the features of the AWS Support Center.

AWSSDK - AWS Storage Gateway
The AWS Storage Gateway is a service connecting an on-premises software appliance with cloud-based storage to pro...

AWSSDK - AWS Security Token Service
The AWS Security Token Service (AWS STS) enables you to provide trusted users with temporary credentials that provide...

AWSSDK - AWS OpsWorks
AWS OpsWorks is an application management service that makes it easy to deploy and operate applications of all shape...

AWSSDK - AWS Lambda
AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute...

Created by: Amazon Web Services
Id: AWSSDK.Core
Version: 3.1.0.0
Last Published: 7/28/2015
Downloads: 1276
License
View License
Project Information
Report Abuse
Description:
The Amazon Web Services SDK for .NET - Core Runtime
Tags: AWS Amazon cloud aws-sdk-v3
Dependencies:
.NETFramework,Version=v4.5
No Dependencies
.NETPortable,Version=v0.0,Profile=net45+win+wpa81+wp80+MonoAndroid10+xamarinios10+MonoTouch10
Microsoft.Net.Http (≥ 2.2.29)
PCLStorage (≥ 1.0.2)
PCLCrypto (≥ 1.0.2.15130)

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Using NuGet from the Package Manager Console

To use NuGet from the Package Manager Console in Visual Studio

- *Visual Studio 2010*

  From the **Tools** menu, choose **Library Package Manager**, and then click **Package Manager Console**.
- *Visual Studio 2012 and later*

  From the **Tools** menu, choose **Nuget Package Manager**, and then click **Package Manager Console**.

You can install the AWSSDK assemblies you want from the Package Manager Console by using the **Install-Package** command. For example, to install the AWSSDK.AutoScaling assembly, use the following command.

```
PM> Install-Package AWSSDK.AutoScaling
```

NuGet also installs any dependencies, such as AWSSDK.Core.

To install an earlier version of a package, use the `-Version` option and specify the package version you want. For example, to install version 3.1.0.0 of the AWS SDK for .NET assembly, use the following command line.

```
PM> Install-Package AWSSDK.Core -Version 3.1.0.0
```

For more information about Package Manager Console commands, see Package Manager Console Commands (v1.3).

Start a New Project

The Toolkit for Visual Studio includes C# project templates for a variety of AWS services, including the following basic templates:

AWS Console Project

A console application that makes basic requests to Amazon S3, Amazon SimpleDB, and Amazon EC2.

AWS Empty Project

A console application that does not include any code.

AWS Web Project

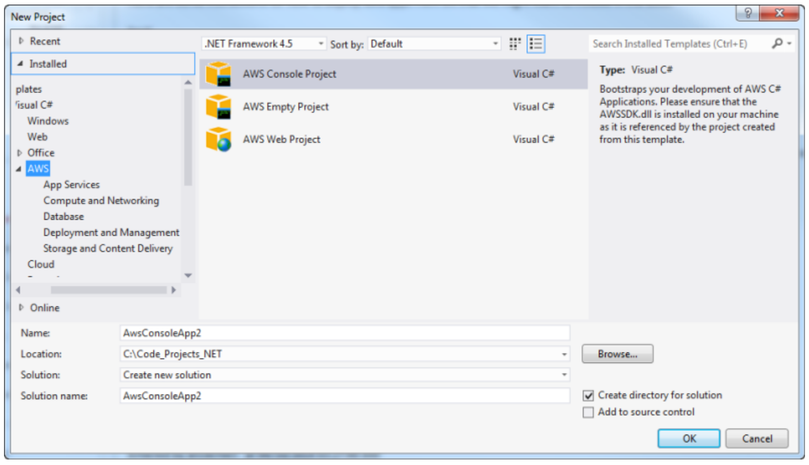
An ASP.NET application that makes basic requests to Amazon S3, Amazon SimpleDB, and Amazon EC2.

You can also base your application on one of the standard Visual Studio project templates. You can add the NuGet packages you need as described in Installing AWSSDK Assemblies with NuGet, or manually add references to the AWSSDK assemblies you need, which are located in the binNet45 subdirectory where the AWS SDK for .NET was installed.

Use the following procedure to get started by creating and running a new AWS Console project for Visual Studio 2012. The process is similar for other project types and Visual Studio versions. For more information about how to configure an AWS application:, see Configuring Your AWS SDK for .NET Application.

To start a new project

1. In Visual Studio, on the **File** menu, choose **New**, and then choose **Project** to open the **New Project** dialog box.
2. Choose **AWS** from the list of installed templates, and then choose the **AWS Console Project** project template. Type a project name, and then click **OK**.



Type: Visual C#

Bootstraps your development of AWS C# Applications. Please ensure that the AWSSDK.dll is installed on your machine as it is referenced by the project created from this template.

Name: AwsConsoleApp2

Location: C:\Code_Projects_NET

Solution: Create new solution

Solution name: AwsConsoleApp2

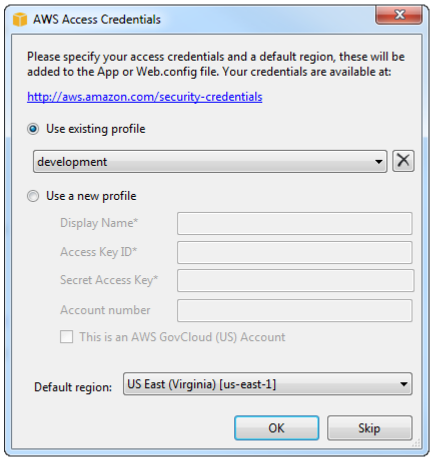
[✓] Create directory for solution

[ ] Add to source control

3. Use the **AWS Access Credentials** dialog box to configure your application.

   • Specify the account profile that your code should use to access AWS. To use an existing profile, choose **Use existing profile**, and then choose the profile from the list. To add a new profile, choose **Use a new profile**, and then type the credentials information. For more information about profiles, see Configuring Your AWS SDK for .NET Application.

- Specify a default AWS Region to use.



Please specify your access credentials and a default region, these will be added to the App or Web.config file. Your credentials are available at:

http://aws.amazon.com/security-credentials

Use existing profile [✓]

development

Use a new profile

Display Name*

Access Key ID*

Secret Access Key*

Account number

This is an AWS GovCloud (US) Account

Default region: US East (Virginia) [us-east-1]

4. Choose **OK** to accept the configuration and open the project. The project's App.config file will contain something similar to the following.

   ```
   <configuration>
     <appSettings>
       <add key="AWSProfileName" value="development"/>
       <add key="AWSRegion" value="us-west-1"/>
     </appSettings>
   </configuration>
   ```

   The Toolkit for Visual Studio puts the values you specified in the **AWS Access Credentials** dialog box into the two key-value pairs in `appSettings`.

   ### Note

   Although using the `appSettings` element is still supported, we recommend you use the `aws` element instead, for example:

   ```
   <configuration>
     <configSections>
       <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
     </configSections>
     <aws region="us-east-1" profileName="development"/>
   </configuration>
   ```

   For more information about the `aws` element, see Configuration Files Reference for AWS SDK for .NET.

5. Choose F5 to compile and run the application, which prints the number of Amazon EC2 instances, Amazon SimpleDB tables, and Amazon S3 buckets in your account.

For more information about configuring an AWS application, see Configuring Your AWS SDK for .NET Application.

Platforms Supported by the AWS SDK for .NET

The AWS SDK for .NET provides distinct groups of assemblies for developers to target different platforms. However, not all SDK functionality is the same on each of these platforms. This topic describes the differences in support for each platform.

.NET Framework 4.5

This version of the AWS SDK for .NET is compiled against .NET Framework 4.5 and runs in the .NET 4.0 runtime. AWS service clients support synchronous and asynhronous calling patterns and use the async and await keywords introduced in C# 5.0.

.NET Framework 3.5

This version of the AWS SDK for .NET is compiled against .NET Framework 3.5, and runs either the .NET 2.0 or .NET 4.0 runtime. AWS service clients support synchronous and asynchronous calling patterns and use the older Begin and End pattern.

Note

The AWS SDK for .NET is not Federal Information Processing Standard (FIPS) compliant when used by applications built against version 2.0 of the CLR. For details on how you can substitute a FIPS compliant implementation in that environment, refer to CryptoConfig on the Microsoft blog and the CLR Security team's HMACSHA256 class (HMACSHA256Cng) in Security.Cryptography.dll.

.NET Core

The AWS SDK for .NET supports applications written for .NET Core. AWS service clients only support asynchronous calling patterns in .NET core. This also affects many of the high level abstractions built on top of service clients like Amazon S3's `TransferUtility` which will only support asynchronous calls in the .NET Core environment. For details, see *Configuring the AWS SDK for .NET with .NET Core*.

Portable Class Library

The AWS SDK for .NET also contains a Portable Class Library implementation. The Portable Class Library implementation can target multiple platforms,including Universal Windows Platform (UWP), and Xamarin on iOS and Android. See the AWS Mobile SDK for .NET and Xamarin for more details. AWS service clients only support asynchronous calling patterns.

Unity Support

The AWS SDK for .NET supports generating Assemblies for Unity. More information can be found in the Unity README.

More Info

- migration-v3

Programming with the AWS SDK for .NET

This section provides general information about developing software with the AWS SDK for .NET.

For information about developing software for specific AWS services, see AWS SDK for .NET Examples.

Configuring Your AWS SDK for .NET Application

You can configure your AWS SDK for .NET application to specify AWS credentials, logging options, endpoints, or signature version 4 support with Amazon S3.

The recommended way to configure an application is to use the `<aws>` element in the project's App.config or Web.config file. The following example specifies the AWSRegion and AWSLogging parameters.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws region="us-west-2">
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Another way to configure an application is to edit the `<appSettings>` element in the project's App.config or Web.config file. The following example specifies the AWSRegion and AWSLogging parameters.

```
<configuration>
  <appSettings>
    <add key="AWSRegion" value="us-west-2"/>
    <add key="AWSLogging" value="log4net"/>
  </appSettings>
</configuration>
```

These settings take effect only after the application has been rebuilt.

Although you can configure an AWS SDK for .NET application programmatically by setting property values in the AWSConfigs class, we recommend you use the `aws` element instead. The following example specifies the AWSRegion and AWSLogging parameters:

```
AWSConfigs.AWSRegion = "us-west-2";
AWSConfigs.Logging = LoggingOptions.Log4Net;
```

Programmatically defined parameters override any values that were specified in an App.config or Web.config file. Some programmatically defined parameter values take effect immediately; others take effect only after you create a new client object. For more information, see Configuring AWS Credentials.

Configuring the AWS SDK for .NET with .NET Core

One of the biggest changes in .NET Core is the removal of `ConfigurationManager` and the standard app.config and web.config files that were used ubiquitously with .NET Framework and ASP.NET applications.

For traditional .NET applications, the AWS SDK for .NET uses this configuration system to set things like AWS credentials and region so that you don't have to do this in code.

The configuration system in .NET Core allows any type of input source from any location. Also, the configuration object isn't a global singleton like the ConfigurationManager in standard .NET applications, so the AWS SDK for .NET doesn't have access to read settings from it.

> **Note**
>
> For background on the .NET Core configuration system, read the Configuration topic in the .NET Core documentation.

To make it easy to use the AWS SDK for .NET with .NET Core, you can use the `AWSSDK.Extensions.NETCore.Setup` NuGet package. Like many .NET Core libraries, it adds extension methods to the `IConfiguration` interface to make getting the AWS configuration seamless.

Using AWSSDK.Extensions.NETCore.Setup

When you create an ASP.NET Core MVC application in Visual Studio, the constructor for Startup.cs handles configuration by reading in various input sources, using the `ConfigurationBuilder` and setting the `Configuration` property to the built `IConfiguration` object.

```
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```

To use the `Configuration` object to get the AWS options, first add the `AWSSDK.Extensions.NETCore.Setup` NuGet package. Then, add your options to the configuration file. Notice one of the files added to the `ConfigurationBuilder` is called `$"appsettings.{env.EnvironmentName}.json"`. If you look at the Debug tab in your project's properties, you can see this file is set to **Development**. This works great for local testing because you can put your configuration in the appsettings.Development.json file, which is read-only during local testing. When you deploy an Amazon EC2 instance that has `EnvironmentName` set to **Production**, this file is ignored and the AWS SDK for .NET falls back to the IAM credentials and region configured for the Amazon EC2 instance.

The configuration below shows an example of the values you can add in the appsettings.Development.json file in your project to supply AWS settings.

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
  }
}
```

To access the AWS options set in the file from code, call the `GetAWSOptions` extension method added on `IConfiguration`. To construct a service client from these options, call `CreateServiceClient`. The following example code shows how to create an Amazon S3 service client.

```
var options = Configuration.GetAWSOptions();
IAmazonS3 client = options.CreateServiceClient<IAmazonS3>();
```

Allowed Values in appsettings File

The following app configuration values can be set in the appsettings.Development.json file. The field names must use the casing shown in the list below. For details on these settings, refer to the AWS.Runtime.ClientConfg class.

- Region
- Profile
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

ASP.NET Core Dependency Injection

The *AWSSDK.Extensions.NETCore.Setup* NuGet package also integrates with a new dependency injection system in ASP.NET Core. The `ConfigureServices` method in `Startup` is where the MVC services are added. If the application is using Entity Framework, this is also where that is initialized.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
}
```

Note

Background on dependency injection in .NET Core is available on the .NET Core documentation site.

The `AWSSDK.Extensions.NETCore.Setup` NuGet package adds new extension methods to `IServiceCollection` that you can use to add AWS services to the dependency injection. The following code shows how to add the AWS options that are read from `IConfiguration` to add Amazon S3 and DynamoDB to our list of services.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    services.AddDefaultAWSOptions(Configuration.GetAWSOptions());
    services.AddAWSService<IAmazonS3>();
    services.AddAWSService<IAmazonDynamoDB>();
}
```

Now, if your MVC controllers use either `IAmazonS3` or `IAmazonDynamoDB` as parameters in their constructors, the dependency injection system passes in those services.

```
public class HomeController : Controller
{
    IAmazonS3 S3Client { get; set; }

    public HomeController(IAmazonS3 s3Client)
    {
        this.S3Client = s3Client;
    }

    ...

}
```

Configuring AWS Credentials

You must manage your AWS credentials securely and avoid practices that can unintentionally expose your credentials to the public. In this topic, we describe how you configure your application's AWS credentials so that they remain secure.

- Don't use your account's root credentials to access your AWS resources. These credentials provide unrestricted account access and are difficult to revoke.
- Don't put literal access keys in your application, including the project's `App.config` or `Web.config` file. If you do, you create a risk of accidentally exposing your credentials if, for example, you upload the project to a public repository.

We assume you have created an AWS account and have access to your credentials. If you haven't yet, see Create an AWS Account and Credentials.

The following are general guidelines for securely managing credentials:

- Create IAM users and use their IAM user credentials instead of using your AWS root user. IAM user credentials are easier to revoke if they're compromised. You can apply a policy to each IAM user that restricts the user to a specific set of resources and actions.
- During application development, the preferred approach for managing credentials is to put a profile for each set of IAM user credentials in the SDK Store. You can also use a plaintext credentials file to store profiles that contain credentials. Then you can reference a specific profile programmatically instead of storing the credentials in your project files. To limit the risk of unintentionally exposing credentials, you should store the SDK Store or credentials file separately from your project files.
- Use IAM roles for applications that are running on Amazon EC2 instances.
- Use temporary credentials or environment variables for applications that are available to users outside your organization.

The following topics describe how to manage credentials for an AWS SDK for .NET application. For a discussion of how to securely manage AWS credentials, see Best Practices for Managing AWS Access Keys.

Using the SDK Store

During development of your AWS SDK for .NET application, add a profile to the SDK Store for each set of credentials you want to use in your application. This prevents the accidental exposure of your AWS credentials. The SDK Store is located in the `C:\Users\<username>\AppData\Local\AWSToolkit` folder in the `RegisteredAccounts.json` file. The SDK Store provides the following benefits:

- The SDK Store can contain multiple profiles from any number of accounts.
- The credentials in the SDK Store are encrypted, and the SDK Store resides in the user's home directory. This limits the risk of accidentally exposing your credentials.
- You reference the profile by name in your application and the associated credentials are referenced at run time. Your source files never contain the credentials.
- If you include a profile named `default`, the AWS SDK for .NET uses that profile. This is also true if you don't provide another profile name, or if the specified name isn't found.
- The SDK Store also provides credentials to AWS Tools for Windows PowerShell and the AWS Toolkit for Visual Studio.

> **Note**
>
> SDK Store profiles are specific to a particular user on a particular host. They cannot be copied to other hosts or other users. For this reason, you cannot use SDK Store profiles in production applications. For more information, see Credential and Profile Resolution.

You can manage the profiles in the SDK Store in several ways.

- Use the graphical user interface (GUI) in the AWS Toolkit for Visual Studio to manage profiles. For more information about adding credentials to the SDK Store by using the GUI, see Specifying Credentials in the AWS Toolkit for Visual Studio.
- You can manage your profiles from the command line by using the `Set-AWSCredentials` cmdlet in AWS Tools for Windows PowerShell. For more information, see Using AWS Credentials.
- You can create and manage your profiles programmatically by using the Amazon.Runtime.CredentialManagement.CredentialProfile class.

The following examples show how to create a basic profile and SAML profile and add them to the SDK Store by using the RegisterProfile method.

Create a Profile and Save it to the .NET Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object and set its `AccessKey` and `SecretKey` properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the `CredentialProfileOptions` object you created. Optionally, set the Region property for the profile. Instantiate a NetSDKCredentialsFile object and call the RegisterProfile method to register the profile.

```
 var options = new CredentialProfileOptions
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};
var profile = new Amazon.Runtime.CredentialManagement.CredentialProfile("basic_profile",
profile.Region = RegionEndpoint.USWest1;
var netSDKFile = new NetSDKCredentialsFile();
netSDKFile.RegisterProfile(profile);
```

The `RegisterProfile` method is used to register a new profile. Your application typically calls this method only once for each profile.

Create a SAMLEndpoint and an Associated Profile and Save it to the .NET Credentials File

Create an Amazon.Runtime.CredentialManagement.SAMLEndpoint object. Provide the name and endpoint URI parameters. Create an Amazon.Runtime.CredentialManagement.SAMLEndpointManager object. Call the RegisterEndpoint method to register the endpoint. Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object and set its `EndpointName` and `RoleArn` properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object and provide the name of the profile and the `CredentialProfileOptions` object you created. Optionally, set the Region property for the profile. Instantiate a NetSDKCredentialsFile object and call the RegisterProfile method to register the profile.

```
var endpoint = new SAMLEndpoint("endpoint1", new Uri("https://some_saml_endpoint"), SAML
var endpointManager = new SAMLEndpointManager();
endpointManager.RegisterEndpoint(endpoint);
options = new CredentialProfileOptions
{
    EndpointName = "endpoint1",
    RoleArn = "arn:aws:iam::999999999999:role/some-role"
};
profile = new CredentialProfile("federated_profile", options);
netSDKFile = new NetSDKCredentialsFile();
netSDKFile.RegisterProfile(profile);
```

Using a Credentials File

You can also store profiles in a shared credentials file. This file can be used by the other AWS SDKs, the AWS CLI and AWS Tools for Windows PowerShell. To reduce the risk of accidentally exposing credentials, store the credentials file separately from any project files, usually in the user's home folder. *Be aware that the profiles in credentials files are stored in plaintext.*

Use can manage the profiles in the shared credentials file in two ways:

- You can a text editor. The file is named credentials, and the default location is under your user's home folder. For example, if your user name is `awsuser`, the credentials file would be C:\users\awsuser\.aws\credentials.

  The following is an example of a profile in the credentials file.

```
   [{profile_name}]
   aws_access_key_id = {accessKey}
   aws_secret_access_key = {secretKey}

 For more information, see
`Best Practices for Managing AWS Access Keys <http://docs.aws.amazon.com/general/latest
```

Tip

If you include a profile named `default`, the AWS SDK for .NET uses that profile by default if it can't find the specified profile.

You can store the credentials file that contains the profiles in a location you choose, such as C:\aws_service_credentials\credentials. You then explicitly specify the file path in the `AWSProfilesLocation` attribute in your project's App.config or Web.config file. For more information, see Specifying a Profile.

- You can programmatically manage the credentials file by using the classes in the Amazon.Runtime.CredentialManagement namespace.

Create a Profile and Save it to the Shared Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object and set its `AccessKey` and `SecretKey` properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the `CredentialProfileOptions` you created. Optionally, set the Region property for the profile. Instantiate an Amazon.Runtime.CredentialManagement.SharedCredentialsFile object and call the RegisterProfile method to register the profile.

```
options = new CredentialProfileOptions
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};
profile = new CredentialProfile("shared_profile", options);
profile.Region = RegionEndpoint.USWest1;
var sharedFile = new SharedCredentialsFile();
sharedFile.RegisterProfile(profile);
```

The `RegisterProfile` method is used to register a new profile. Your application will normally call this method only once for each profile.

Create a Source Profile and an Associated Assume Role Profile and Save It to the Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object for the source profile and set its `AccessKey` and `SecretKey` properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the `CredentialProfileOptions` you created. Instantiate an Amazon.Runtime.CredentialManagement.SharedCredentialsFile object and call the RegisterProfile method to register the profile. Create another Amazon.Runtime.CredentialManagement.CredentialProfileOptions object for the assumed role profile and set the `SourceProfile` and `RoleArn` properties for the profile. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object for the assumed role. Provide the name of the profile and the `CredentialProfileOptions` you created.

```
// Create the source profile and save it to the shared credentials file
var sourceProfileOptions = new CredentialProfileOptions
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};
var sourceProfile = new CredentialProfile("source_profile", sourceProfileOptions);
sharedFile = new SharedCredentialsFile();
sharedFile.RegisterProfile(sourceProfile);

// Create the assume role profile and save it to the shared credentials file
var assumeRoleProfileOptions = new CredentialProfileOptions
{
    SourceProfile = "source_profile",
    RoleArn = "arn:aws:iam::999999999999:role/some-role"
};
var assumeRoleProfile = new CredentialProfile("assume_role_profile", assumeRoleProfileOp
sharedFile.RegisterProfile(assumeRoleProfile);
```

Update an Existing Profile in the Shared Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Set the `Region`, `AccessKey` and `SecretKey` properties for the profile. Call the TryGetProfile method. If the profile exists, use an Amazon.Runtime.CredentialManagement.SharedCredentialsFile instance to call the RegisterProfile method to register the updated profile.

```
sharedFile = new SharedCredentialsFile();
CredentialProfile basicProfile;
if (sharedFile.TryGetProfile("basicProfile", out basicProfile))
{
    basicProfile.Region = RegionEndpoint.USEast1;
    basicProfile.Options.AccessKey = "different_access_key";
    basicProfile.Options.SecretKey = "different_secret_key";

    sharedFile.RegisterProfile(basicProfile);
}
```

Accessing Credentials and Profiles in an Application

You can easily locate credentials and profiles in the .NET credentials file or in the shared credentials file by using the Amazon.Runtime.CredentialManagement.CredentialProfileStoreChain class. This is the way the .NET SDK looks for credentials and profiles. The `CredentialProfileStoreChain` class automatically checks in both credentials files.

You can get credentials or profiles by using the TryGetAWSCredentials or TryGetProfile methods. The `ProfilesLocation` property determines the behavior of the `CredentialsProfileChain`, as follows:

1. If ProfilesLocation is non-null and non-empty, search the shared credentials file at the disk path in the `ProfilesLocation` property.
2. If `ProfilesLocation` is null or empty and the platform supports the .NET credentials file, search the .NET credentials file. If the profile is not found, search the shared credentials file in the default location.
3. If `ProfilesLocation` is null or empty and the platform doesn't support the .NET credentials file, search the shared credentials file in the default location.

Get Credentials from the SDK Credentials File or the Shared Credentials File in the Default Location.

Create a `CredentialProfileStoreChain` object and an Amazon.Runtime.AWSCredentials object. Call the `TryGetAWSCredentials` method. Provide the profile name and the `AWSCredentials` object in which to return the credentials.

```
var chain = new CredentialProfileStoreChain();
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // use awsCredentials
}
```

Get a Profile from the SDK Credentials File or the Shared Credentials File in the Default Location

Create a `CredentialProfileStoreChain` object and an Amazon.Runtime.CredentialManagement.CredentialProfile object. Call the `TryGetProfile` method and provide the profile name and `CredentialProfile` object in which to return the credentials.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Get AWSCredentials from a File in the Shared Credentials File Format at a File Location

Create a `CredentialProfileStoreChain` object and provide the path to the credentials file. Create an `AWSCredentials` object. Call the `TryGetAWSCredentials` method. Provide the profile name and the `AWSCredentials` object in which to return the credentials.

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
```

```
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials
}
```

How to Create an AmazonS3Client Using the SharedCredentialsFile Class

You can create an AmazonS3Client object that uses the credentials for a specific profile by using the Amazon.Runtime.CredentialManagement.SharedCredentialsFile class. The AWS SDK for .NET loads the credentials contained in the profile automatically. You might do this if you want to use a specific profile for a given client that is different from the `profile` you specify in `App.Config`.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out awsCredentials
{
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = client.ListBuckets();
    }
}
```

If you want to use the default profile, and have the AWS SDK for .NET automatically use your default credentials to create the client object use the following code.

```
using (var client = new AmazonS3Client(RegionEndpoint.US-West2))
{
    var response = client.ListBuckets();
}
```

Credential and Profile Resolution

The AWS SDK for .NET searches for credentials in the following order and uses the first available set for the current application.

1. The client configuration, or what is explicitly set on the AWS service client.
2. `BasicAWSCredentials` that are created from the `AWSAccessKey` and `AWSSecretKey` `AppConfig` values, if they're available.
3. A credentials profile with the name specified by a value in `AWSConfigs.AWSProfileName` (set explicitly or in `AppConfig`).
4. The :code"*default* credentials profile.
5. `SessionAWSCredentials` that are created from the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables, if they're all non-empty.

6. `BasicAWSCredentials` that are created from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables, if they're both non-empty.
7. EC2 instance metadata.

SDK Store profiles are specific to a particular user on a particular host. You can't copy them to other hosts or other users. For this reason, you can't reuse SDK Store profiles that are on your development machine on other hosts or developer machines. If your application is running on an Amazon EC2 instance, use an IAM role as described in Using IAM Roles for EC2 Instances with the AWS SDK for .NET. Otherwise, store your credentials in a credentials file that your web application has access to on the server.

Profile Resolution

With two different credentials file types, it's important to understand how to configure the AWS SDK for .NET and AWS Tools for Windows PowerShell to use them. The `AWSConfigs.AWSProfilesLocation` (set explicitly or in `AppConfig`) controls how the AWS SDK for .NET finds credential profiles. The `-ProfileLocation` command line argument controls how AWS Tools for Windows PowerShell finds a profile. Here's how the configuration works in both cases.

Profile Location Value	Profile Resolution Behavior
null (not set) or empty	First search the .NET credentials file for a profile with the specified name. If the profile isn't there, search `%HOME%\.aws\credentials`. If the profile isn't there, search `%HOME%\.aws\config`.
The path to a file in the shared credentials file format	Search *only* the specified file for a profile with the specified name.

Specifying a Profile

Profiles are the preferred way to use credentials in an AWS SDK for .NET application. You don't have to specify where the profile is stored. You only reference the profile by name. The AWS SDK for .NET retrieves the corresponding credentials, as described in the previous section.

The preferred way to specify a profile is to define an `AWSProfileName` value in the `appSettings` section of your application's App.config or Web.config file. The associated credentials are incorporated into the application during the build process.

The following example specifies a profile named `development`.

```
<configuration>
  <appSettings>
    <add key="AWSProfileName" value="development"/>
  </appSettings>
</configuration>
```

This example assumes the profile exists in the SDK Store or in a credentials file in the default location.

If your profiles are stored in a credentials file in another location, specify the location by adding a `AWSProfilesLocation` attribute value in the `<appSettings>` element. The following example specifies C:\aws_service_credentials\credentials as the credentials file.

```
<configuration>
  <appSettings>
```

```
    <add key="AWSProfileName" value="development"/>
    <add key="AWSProfilesLocation" value="C:\aws_service_credentials\credentials"/>
  </appSettings>
</configuration>
```

The deprecated alternative way to specify a profile is shown below for completeness, but we do not recommend it.

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws profileName="development" profilesLocation="C:\aws_service_credentials\credentials"/
</configuration>

<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection,AWSSDK.Core"/>
  </configSections>
  <aws profileName="development" profilesLocation="C:\aws_service_credentials\credentials"/
</configuration>
```

Using Federated User Account Credentials

Applications that use the AWS SDK for .NET (AWSSDK.Core version 3.1.6.0 and later) can use federated user accounts through Active Directory Federation Services (AD FS) to access AWS web services by using Security Assertion Markup Language (SAML).

Federated access support means users can authenticate using your Active Directory. Temporary credentials are granted to the user automatically. These temporary credentials, which are valid for one hour, are used when your application invokes AWS web services. The SDK handles management of the temporary credentials. For domain-joined user accounts, if your application makes a call but the credentials have expired, the user is reauthenticated automatically and fresh credentials are granted. (For non-domain-joined accounts, the user is prompted to enter credentials before reauthentication.)

To use this support in your .NET application, you must first set up the role profile by using a PowerShell cmdlet. To learn how, see the AWS Tools for Windows PowerShell documentation.

After you setup the role profile, reference the profile in your application's app.config/web.config file with the `AWSProfileName` key in the same way you would with other credential profiles.

The SDK Security Token Service assembly (AWSSDK.SecurityToken.dll), which is loaded at runtime, provides the SAML support to obtain AWS credentials. Be sure this assembly is available to your application at run time.

Specifying Roles or Temporary Credentials

For applications that run on Amazon EC2 instances, the most secure way to manage credentials is to use IAM roles, as described in Using IAM Roles for EC2 Instances with the AWS SDK for .NET.

For application scenarios in which the software executable is available to users outside your organization, we recommend you design the software to use *temporary security credentials*. In addition to providing restricted access to AWS resources, these credentials have the benefit of expiring after a specified period of time. For more information about temporary security credentials, see the following:

- Using Security Tokens to Grant Temporary Access to Your AWS Resources
- Authenticating Users of AWS Mobile Applications with a Token Vending Machine.

Although the title of the second article refers specifically to mobile applications, the article contains information that is useful for any AWS application deployed outside of your organization.

Using Proxy Credentials

If your software communicates with AWS through a proxy, you can specify credentials for the proxy by using the `ProxyCredentials` property on the ClientConfig class for the service. For example, for Amazon S3 you could use code similar to the following, where {my-username} and {my-password} are the proxy user name and password specified in a NetworkCredential object.

```
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential("my-username", "my-password");
```

Earlier versions of the SDK used `ProxyUsername` and `ProxyPassword`, but these properties are deprecated.

AWS Region Selection

AWS Regions allow you to access AWS services that physically reside in a specific geographic region. This can be useful for redundancy and to keep your data and applications running close to where you and your users will access them. You can specify a region when creating the AWS service client by using the RegionEndpoint class.

Here is an example that instantiates an Amazon EC2 client in a specific region.

```
AmazonEC2Client ec2Client = new AmazonEC2Client(RegionEndpoint.USEast1);
```

Regions are isolated from each other. For example, you can't access US East (N. Virginia) resources when using the EU (Ireland) region. If your code needs access to multiple AWS Regions, we recommend you create a separate client for each region.

To use services in the China (Beijing) Region, you must have an account and credentials that are specific to the China (Beijing) Region. Accounts and credentials for other AWS Regions won't work for the China (Beijing) Region. Likewise, accounts and credentials for the China (Beijing) Region won't work for other AWS Regions. For information about endpoints and protocols that are available in the China (Beijing) Region, see China (Beijing) Region.

New AWS services can be launched initially in a few regions and then supported in other regions. In these cases you don't need to install the latest SDK to access the new regions. You can specify newly added regions on a per-client basis or globally.

Per-Client

Construct the new region endpoint by using GetBySystemName:

```
var newRegion = RegionEndpoint.GetBySystemName("us-west-new");
using (var ec2Client = new AmazonEC2Client(newRegion))
{
  // Make a request to EC2 using ec2Client
}
```

You can also use the `ServiceURL` property of the service client configuration class to specify the region. This technique works even if the region endpoint does not follow the regular region endpoint pattern.

```
var ec2ClientConfig = new AmazonEC2Config
{
    // Specify the endpoint explicitly
    ServiceURL = "https://ec2.us-west-new.amazonaws.com"
};

using (var ec2Client = new AmazonEC2Client(newRegion))
{
  // Make a request to EC2 using ec2Client
}
```

Globally

You can set the region globally in three ways.

You can set the `AWSConfigs.AWSRegion` property,

```
AWSConfigs.AWSRegion = "us-west-new";
using (var ec2Client = new AmazonEC2Client())
{
  // Make request to Amazon EC2 using ec2Client
}
```

You can set the `AWSRegion` key in the `appSettings` section of the `app.config` file.

```
<configuration>
  <appSettings>
    <add key="AWSRegion" value="us-west-2"/>
  </appSettings>
</configuration>
```

You can set the `region` attribute in the `aws` section as described in AWSRegion.

```
<aws region="us-west-2"/>
```

To view the current list of all supported regions and endpoints for each AWS service, see Regions and Endpoints in the *Amazon Web Services General Reference*.

Configuring Other Application Parameters

In addition to configuring credentials, you can configure a number of other application parameters:

- AWSLogging
- AWSLogMetrics
- AWSRegion
- AWSResponseLogging
- AWS.DynamoDBContext.TableNamePrefix
- AWS.S3.UseSignatureVersion4
- AWSEndpointDefinition
- AWS Service-Generated Endpoints

These parameters can be configured in the application's App.config or Web.config file. Although you can also configure these with the AWS SDK for .NET API, we recommend you use the application's .config file. Both approaches are described here.

For more information about use of the `<aws>` element as described later in this topic, see Configuration Files Reference for AWS SDK for .NET.

AWSLogging

Configures how the SDK should log events, if at all. For example, the recommended approach is to use the `<logging>` element, which is a child element of the `<aws>` element:

```
<aws>
  <logging logTo="Log4Net"/>
</aws>
```

Alternatively:

```
<add key="AWSLogging" value="log4net"/>
```

The possible values are:

`None`
: Turn off event logging. This is the default.

`log4net`
: Log using log4net.

`SystemDiagnostics`
: Log using the `System.Diagnostics` class.

You can set multiple values for the `logTo` attribute, separated by commas. The following example sets both `log4net` and `System.Diagnostics` logging in the .config file:

```
<logging logTo="Log4Net, SystemDiagnostics"/>
```

Alternatively:

```
<add key="AWSLogging" value="log4net, SystemDiagnostics"/>
```

Alternatively, using the AWS SDK for .NET API, combine the values of the LoggingOptions enumeration and set the AWSConfigs.Logging property:

```
AWSConfigs.Logging = LoggingOptions.Log4Net | LoggingOptions.SystemDiagnostics;
```

Changes to this setting take effect only for new AWS client instances.

AWSLogMetrics

Specifies whether or not the SDK should log performance metrics. To set the metrics logging configuration in the .config file, set the `logMetrics` attribute value in the `<logging>` element, which is a child element of the `<aws>` element:

```
<aws>
  <logging logMetrics="true"/>
</aws>
```

Alternatively, set the `AWSLogMetrics` key in the `<appSettings>` section:

```
<add key="AWSLogMetrics" value="true">
```

Alternatively, to set metrics logging with the AWS SDK for .NET API, set the AWSConfigs.LogMetrics property:

```
AWSConfigs.LogMetrics = true;
```

This setting configures the default `LogMetrics` property for all clients/configs. Changes to this setting take effect only for new AWS client instances.

AWSRegion

Configures the default AWS region for clients that have not explicitly specified a region. To set the region in the .config file, the recommended approach is to set the `region` attribute value in the `aws` element:

```
<aws region="us-west-2"/>
```

Alternatively, set the *AWSRegion* key in the `<appSettings>` section:

```
<add key="AWSRegion" value="us-west-2"/>
```

Alternatively, to set the region with the AWS SDK for .NET API, set the AWSConfigs.AWSRegion property:

```
AWSConfigs.AWSRegion = "us-west-2";
```

For more information about creating an AWS client for a specific region, see AWS Region Selection. Changes to this setting take effect only for new AWS client instances.

AWSResponseLogging

Configures when the SDK should log service responses. The possible values are:

`Never`
: Never log service responses. This is the default.

`Always`
: Always log service responses.

`OnError`
: Only log service responses when an error occurs.

To set the service logging configuration in the .config file, the recommended approach is to set the `logResponses` attribute value in the `<logging>` element, which is a child element of the `<aws>` element:

```
<aws>
  <logging logResponses="OnError"/>
</aws>
```

Alternatively, set the *AWSResponseLogging* key in the `<appSettings>` section:

```
<add key="AWSResponseLogging" value="OnError"/>
```

Alternatively, to set service logging with the AWS SDK for .NET API, set the AWSConfigs.ResponseLogging property to one of the values of the ResponseLoggingOption enumeration:

```
AWSConfigs.ResponseLogging = ResponseLoggingOption.OnError;
```

Changes to this setting take effect immediately.

AWS.DynamoDBContext.TableNamePrefix

Configures the default `TableNamePrefix` the `DynamoDBContext` will use if not manually configured.

To set the table name prefix in the .config file, the recommended approach is to set the `tableNamePrefix` attribute value in the `<dynamoDBContext>` element, which is a child element of the `<dynamoDB>` element, which itself is a child element of the `<aws>` element:

```
<dynamoDBContext tableNamePrefix="Test-"/>
```

Alternatively, set the `AWS.DynamoDBContext.TableNamePrefix` key in the `<appSettings>` section:

```
<add key="AWS.DynamoDBContext.TableNamePrefix" value="Test-"/>
```

Alternatively, to set the table name prefix with the AWS SDK for .NET API, set the AWSConfigs.DynamoDBContextTableNamePrefix property:

```
AWSConfigs.DynamoDBContextTableNamePrefix = "Test-";
```

Changes to this setting will take effect only in newly constructed instances of `DynamoDBContextConfig` and `DynamoDBContext`.

AWS.S3.UseSignatureVersion4

Configures whether or not the Amazon S3 client should use signature version 4 signing with requests.

To set signature version 4 signing for Amazon S3 in the .config file, the recommended approach is to set the `useSignatureVersion4` attribute of the `<s3>` element, which is a child element of the `<aws>` element:

```
<aws>
  <s3 useSignatureVersion4="true"/>
</aws>
```

Alternatively, set the *AWS.S3.UseSignatureVersion4* key to *true* in the `<appSettings>` section:

```
<add key="AWS.S3.UseSignatureVersion4" value="true"/>
```

Alternatively, to set signature version 4 signing with the AWS SDK for .NET API, set the AWSConfigs.S3UseSignatureVersion4 property to `true`:

```
AWSConfigs.S3UseSignatureVersion4 = true;
```

By default, this setting is `false`, but signature version 4 may be used by default in some cases or with some regions. When the setting is `true`, signature version 4 will be used for all requests. Changes to this setting take effect only for new Amazon S3 client instances.

AWSEndpointDefinition

Configures whether the SDK should use a custom configuration file that defines the regions and endpoints.

To set the endpoint definition file in the .config file, we recommend setting the `endpointDefinition` attribute value in the `<aws>` element.

```
<aws endpointDefinition="c:\config\endpoints.json"/>
```

Alternatively, you can set the *AWSEndpointDefinition* key in the `<appSettings>` section:

```
<add key="AWSEndpointDefinition" value="c:\config\endpoints.json"/>
```

Alternatively, to set the endpoint definition file with the AWS SDK for .NET API, set the AWSConfigs.EndpointDefinition property:

```
AWSConfigs.EndpointDefinition = @"c:\config\endpoints.json";
```

If no file name is provided, then a custom configuration file will not be used. Changes to this setting take effect only for new AWS client instances. The endpoint.json file is available from https://github.com/aws/aws-sdk-net/blob/master/sdk/src/Core/endpoints.json.

AWS Service-Generated Endpoints

Some AWS services generate their own endpoints instead of consuming a region endpoint. Clients for these services consume a service Url that is specific to that service and your resources. Two examples of these services are Amazon CloudSearch and AWS IoT. The following examples show how you can obtain the endpoints for those services.

Amazon CloudSearch Endpoints Example

The Amazon CloudSearch client is used for accessing the Amazon CloudSearch configuration service. You use the Amazon CloudSearch configuration service to create, configure, and manage search domains. To create a search domain, create a CreateDomainRequest object and provide the `DomainName` property. Create an AmazonCloudSearchClient object by using the request object. Call the CreateDomain method. The CreateDomainResponse object returned from the call contains a `DomainStatus` property that has both the `DocService` and `SearchService` endpoints. Create an AmazonCloudSearchDomainConfig object and use it to initialize `DocService` and `SearchService` instances of the AmazonCloudSearchDomainClient class.

```
// Create domain and retrieve DocService and SearchService endpoints
DomainStatus domainStatus;
using (var searchClient = new AmazonCloudSearchClient())
{
    var request = new CreateDomainRequest
    {
        DomainName = "testdomain"
    };
    domainStatus = searchClient.CreateDomain(request).DomainStatus;
    Console.WriteLine(domainStatus.DomainName + " created");
}

// Test the DocService endpoint
var docServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.DocService.Endpoint
};
using (var domainDocService = new AmazonCloudSearchDomainClient(docServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain DocService client instantiated using the Do
    Console.WriteLine("DocService endpoint = " + domainStatus.DocService.Endpoint);

    using (var docStream = new FileStream(@"C:\doc_source\XMLFile4.xml", FileMode.Open))
    {
        var upload = new UploadDocumentsRequest
        {
            ContentType = ContentType.ApplicationXml,
            Documents = docStream
        };
        domainDocService.UploadDocuments(upload);
    }
}

// Test the SearchService endpoint
var searchServiceConfig = new AmazonCloudSearchDomainConfig
```

```
{
    ServiceURL = "https://" + domainStatus.SearchService.Endpoint
};
using (var domainSearchService = new AmazonCloudSearchDomainClient(searchServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain SearchService client instantiated using the
    Console.WriteLine("SearchService endpoint = " + domainStatus.SearchService.Endpoint);

    var searchReq = new SearchRequest
    {
        Query = "Gambardella",
        Sort = "_score desc",
        QueryParser = QueryParser.Simple
    };
    var searchResp = domainSearchService.Search(searchReq);
}
```

AWS IoT Endpoints Example

To obtain the endpoint for AWS IoT, create an AmazonIoTClient object and call the DescribeEndPoint method. The returned DescribeEndPointResponse object contains the `EndpointAddress`. Create an AmazonIotDataConfig object, set the `ServiceURL` property, and use the object to instantiate the AmazonIotDataClient class.

```
string iotEndpointAddress;
using (var iotClient = new AmazonIoTClient())
{
    var endPointResponse = iotClient.DescribeEndpoint();
    iotEndpointAddress = endPointResponse.EndpointAddress;
}

var ioTdocServiceConfig = new AmazonIotDataConfig
{
    ServiceURL = "https://" + iotEndpointAddress
};
using (var dataClient = new AmazonIotDataClient(ioTdocServiceConfig))
{
    Console.WriteLine("AWS IoTData client instantiated using the endpoint from the IotClier
}nstantiated using the endpoint from the IoT client");
```

Configuration Files Reference for AWS SDK for .NET

You can use a .NET project's App.config or Web.config file to specify AWS settings, such as AWS credentials, logging options, AWS service endpoints, and AWS regions, as well as some settings for AWS services, such as Amazon DynamoDB, Amazon EC2, and Amazon S3. The following information describes how to properly format an App.config or Web.config file to specify these types of settings.

Note

Although you can continue to use the `<appSettings>` element in an App.config or Web.config file to specify AWS settings, we recommend you use the `<configSections>` and `<aws>` elements as described later in this topic. For more information about the `<appSettings>` element, see the `<appSettings>` element examples in Configuring Your AWS SDK for .NET Application.

Note

Although you can continue to use the following AWSConfigs class properties in a code file to specify AWS settings, the following properties are deprecated and may not be supported in future releases:

- `DynamoDBContextTableNamePrefix`
- `EC2UseSignatureVersion4`
- `LoggingOptions`
- `LogMetrics`
- `ResponseLoggingOption`
- `S3UseSignatureVersion4`

In general, we recommend that instead of using `AWSConfigs` class properties in a code file to specify AWS settings, you should use the `<configSections>` and `<aws>` elements in an App.config or Web.config file to specify AWS settings, as described later in this topic. For more information about the preceding properties, see the `AWSConfigs` code examples in Configuring Your AWS SDK for .NET Application.

Declaring an AWS Settings Section

You specify AWS settings in an App.config or Web.config file from within the `<aws>` element. Before you can begin using the `<aws>` element, you must create a `<section>` element (which is a child element of the `<configSections>` element) and set its `name` attribute to `aws` and its `type` attribute to `Amazon.AWSSection, AWSSDK.Core`, as shown in the following example:

```
<?xml version="1.0"?>
<configuration>
  ...
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <!-- Add your desired AWS settings declarations here. -->
  </aws>
  ...
</configuration>
```

The Visual Studio Editor does not provide automatic code completion (IntelliSense) for the `<aws>` element or its child elements.

To assist you in creating a correctly formatted version of the `<aws>` element, call the `Amazon.AWSConfigs.GenerateConfigTemplate` method. This outputs a canonical version of the `<aws>` element as a pretty-printed string, which you can adapt to your needs. The following sections describe the `<aws>` element's attributes and child elements.

Allowed Elements

The following is a list of the logical relationships among the allowed elements in an AWS settings section. You can generate the latest version of this list by calling the `Amazon.AWSConfigs.GenerateConfigTemplate` method, which outputs a canonical version of the `<aws>` element as a string you can adapt to your needs.

```
<aws
  endpointDefinition="string value"
  region="string value"
  profileName="string value"
  profilesLocation="string value">
  <logging
    logTo="None, Log4Net, SystemDiagnostics"
    logResponses="Never | OnError | Always"
    logMetrics="true | false"
    logMetricsFormat="Standard | JSON"
    logMetricsCustomFormatter="NameSpace.Class, Assembly" />
  <dynamoDB
    conversionSchema="V1 | V2">
    <dynamoDBContext
      tableNamePrefix="string value">
      <tableAliases>
        <alias
          fromTable="string value"
          toTable="string value" />
      </tableAliases>
      <map
        type="NameSpace.Class, Assembly"
        targetTable="string value">
        <property
          name="string value"
          attribute="string value"
          ignore="true | false"
          version="true | false"
          converter="NameSpace.Class, Assembly" />
      </map>
    </dynamoDBContext>
  </dynamoDB>
  <s3
    useSignatureVersion4="true | false" />
  <ec2
    useSignatureVersion4="true | false" />
  <proxy
    host="string value"
    port="1234"
    username="string value"
    password="string value" />
</aws>
```

Elements Reference

The following is a list of the elements that are allowed in an AWS settings section. For each element, its allowed attributes and parent-child elements are listed.

alias

The `<alias>` element represents a single item in a collection of one or more from-table to to-table mappings that specifies a different table than one configured for a type. This element maps to an instance of the `Amazon.Util.TableAlias` class from the `Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases` property in the AWS SDK for .NET. Remapping is done before applying a table name prefix.

This element can include the following attributes:

`fromTable`

The from-table portion of the from-table to to-table mapping. This attribute maps to the `Amazon.Util.TableAlias.FromTable` property in the AWS SDK for .NET.

`toTable`

The to-table portion of the from-table to to-table mapping. This attribute maps to the `Amazon.Util.TableAlias.ToTable` property in the AWS SDK for .NET.

The parent of the `<alias>` element is the `<tableAliases>` element.

The `<alias>` element contains no child elements.

The following is an example of the `<alias>` element in use:

```
<alias
  fromTable="Studio"
  toTable="Studios" />
```

aws

The `<aws>` element represents the top-most element in an AWS settings section. This element can include the following attributes:

`endpointDefinition`

The absolute path to a custom configuration file that defines the AWS regions and endpoints to use. This attribute maps to the `Amazon.AWSConfigs.EndpointDefinition` property in the AWS SDK for .NET.

`profileName`

The profile name for stored AWS credentials that will be used to make service calls. This attribute maps to the `Amazon.AWSConfigs.AWSProfileName` property in the AWS SDK for .NET.

`profilesLocation`

The absolute path to the location of the credentials file shared with other AWS SDKs. By default, the credentials file is stored in the .aws directory in the current user's home directory. This attribute maps to the `Amazon.AWSConfigs.AWSProfilesLocation` property in the AWS SDK for .NET.

`region`

The default AWS region ID for clients that have not explicitly specified a region. This attribute maps to the `Amazon.AWSConfigs.AWSRegion` property in the AWS SDK for .NET.

The `<aws>` element has no parent element.

The `<aws>` element can include the following child elements:

- `<dynamoDB>`
- `<ec2>`
- `<logging>`
- `<proxy>`
- `<s3>`

The following is an example of the `<aws>` element in use:

```
<aws
  endpointDefinition="C:\Configs\endpoints.xml"
  region="us-west-2"
  profileName="development"
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

The `<dynamoDB>` element represents a collection of settings for Amazon DynamoDB. This element can include the *conversionSchema* attribute, which represents the version to use for converting between .NET and DynamoDB objects. Allowed values include V1 and V2. This attribute maps to the `Amazon.DynamoDBv2.DynamoDBEntryConversion` class in the AWS SDK for .NET. For more information, see DynamoDB Series - Conversion Schemas.

The parent of the `<dynamoDB>` element is the `<aws>` element.

The `<dynamoDB>` element can include the `<dynamoDBContext>` child element.

The following is an example of the `<dynamoDB>` element in use:

```
<dynamoDB
  conversionSchema="V2">
  <!-- ... -->
</dynamoDB>
```

dynamoDBContext

The `<dynamoDBContext>` element represents a collection of Amazon DynamoDB context-specific settings. This element can include the *tableNamePrefix* attribute, which represents the default table name prefix the DynamoDB context will use if it is not manually configured. This attribute maps to the `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` property from the `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` property in the AWS SDK for .NET. For more information, see Enhancements to the DynamoDB SDK.

The parent of the `<dynamoDBContext>` element is the `<dynamoDB>` element.

The `<dynamoDBContext>` element can include the following child elements:

- `<alias>` (one or more instances)
- `<map>` (one or more instances)

The following is an example of the `<dynamoDBContext>` element in use:

```
<dynamoDBContext
  tableNamePrefix="Test-">
  <!-- ... -->
</dynamoDBContext>
```

ec2

The `<ec2>` element represents a collection of Amazon EC2 settings. This element can include the *useSignatureVersion4* attribute, which specifies whether signature version 4 signing will be used for all requests (true) or whether signature version 4 signing will not be used for all requests (false, the default). This attribute maps the `Amazon.Util.EC2Config.UseSignatureVersion4` property from the `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` property in the AWS SDK for .NET.

The parent of the `<ec2>` element is the element.

The `<ec2>` element contains no child elements.

The following is an example of the `<ec2>` element in use:

```
<ec2
  useSignatureVersion4="true" />
```

logging

The `<logging>` element represents a collection of settings for response logging and performance metrics logging. This element can include the following attributes:

`logMetrics`

Whether performance metrics will be logged for all clients and configurations (true); otherwise, false. This attribute maps to the `Amazon.Util.LoggingConfig.LogMetrics` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetrics` property in the AWS SDK for .NET.

`logMetricsCustomFormatter`

The data type and assembly name of a custom formatter for logging metrics. This attribute maps to the `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` property in the AWS SDK for .NET.

`logMetricsFormat`

The format in which the logging metrics are presented (maps to the `Amazon.Util.LoggingConfig.LogMetricsFormat` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` property in the AWS SDK for .NET).

Allowed values include:

`JSON`

Use JSON format.

`Standard`

Use the default format.

`logResponses`

When to log service responses (maps to the `Amazon.Util.LoggingConfig.LogResponses` property from the `Amazon.AWSConfigs.LoggingConfig.LogResponses` property in the AWS SDK for .NET).

Allowed values include:

`Always`

Always log service responses.

`Never`

Never log service responses.

`OnError`

Only log service responses when there are errors.

`logTo`

Where to log to (maps to the `LogTo` property from the `Amazon.AWSConfigs.LoggingConfig.LogTo` property in the AWS SDK for .NET).

Allowed values include one or more of:

`Log4Net`

Log to log4net.

`None`

Disable logging.

`SystemDiagnostics`

Log to System.Diagnostics.

The parent of the `<logging>` element is the `<aws>` element.

The `<logging>` element contains no child elements.

The following is an example of the `<logging>` element in use:

```
<logging
  logTo="SystemDiagnostics"
  logResponses="OnError"
  logMetrics="true"
  logMetricsFormat="JSON"
  logMetricsCustomFormatter="MyLib.Util.MyMetricsFormatter, MyLib" />
```

map

The `<map>` element represents a single item in a collection of type-to-table mappings from .NET types to DynamoDB tables (maps to an instance of the `TypeMapping` class from the `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` property in the AWS SDK for .NET). For more information, see Enhancements to the DynamoDB SDK.

This element can include the following attributes:

`targetTable`

The DynamoDB table to which the mapping applies. This attribute maps to the `Amazon.Util.TypeMapping.TargetTable` property in the AWS SDK for .NET.

`type`

The type and assembly name to which the mapping applies. This attribute maps to the `Amazon.Util.TypeMapping.Type` property in the AWS SDK for .NET.

The parent of the `<map>` element is the `<dynamoDBContext>` element.

The `<map>` element can include one or more instances of the `<property>` child element.

The following is an example of the `<map>` element in use:

```
<map
  type="SampleApp.Models.Movie, SampleDLL"
  targetTable="Movies">
  <!-- ... -->
</map>
```

property

The `<property>` element represents a DynamoDB property. (This element maps to an instance of the Amazon.Util.PropertyConfig class from the `AddProperty` method in the AWS SDK for .NET) For more information, see Enhancements to the DynamoDB SDK and DynamoDB Attributes.

This element can include the following attributes:

`attribute`

The name of an attribute for the property, such as the name of a range key. This attribute maps to the `Amazon.Util.PropertyConfig.Attribute` property in the AWS SDK for .NET.

`converter`

The type of converter that should be used for this property. This attribute maps to the `Amazon.Util.PropertyConfig.Converter` property in the AWS SDK for .NET.

`ignore`

Whether the associated property should be ignored (true); otherwise, false. This attribute maps to the `Amazon.Util.PropertyConfig.Ignore` property in the AWS SDK for .NET.

`name`

The name of the property. This attribute maps to the `Amazon.Util.PropertyConfig.Name` property in the AWS SDK for .NET.

`version`

Whether this property should store the item version number (true); otherwise, false. This attribute maps to the `Amazon.Util.PropertyConfig.Version` property in the AWS SDK for .NET.

The parent of the `<property>` element is the `<map>` element.

The `<property>` element contains no child elements.

The following is an example of the `<property>` element in use:

```
<property
  name="Rating"
  converter="SampleApp.Models.RatingConverter, SampleDLL" />
```

proxy

The `<proxy>` element represents settings for configuring a proxy for the AWS SDK for .NET to use. This element can include the following attributes:

host

The host name or IP address of the proxy server. This attributes maps to the `Amazon.Util.ProxyConfig.Host` property from the `Amazon.AWSConfigs.ProxyConfig.Host` property in the AWS SDK for .NET.

password

The password to authenticate with the proxy server. This attributes maps to the `Amazon.Util.ProxyConfig.Password` property from the `Amazon.AWSConfigs.ProxyConfig.Password` property in the AWS SDK for .NET.

port

The port number of the proxy. This attributes maps to the `Amazon.Util.ProxyConfig.Port` property from the `Amazon.AWSConfigs.ProxyConfig.Port` property in the AWS SDK for .NET.

username

The user name to authenticate with the proxy server. This attributes maps to the `Amazon.Util.ProxyConfig.Username` property from the `Amazon.AWSConfigs.ProxyConfig.Username` property in the AWS SDK for .NET.

The parent of the `<proxy>` element is the `<aws>` element.

The `<proxy>` element contains no child elements.

The following is an example of the `<proxy>` element in use:

```
<proxy
  host="192.0.2.0"
  port="1234"
  username="My-Username-Here"
  password="My-Password-Here" />
```

s3

The `<s3>` element represents a collection of Amazon S3 settings. This element can include the *useSignatureVersion4* attribute, which specifies whether signature version 4 signing will be used for all requests (true) or whether signature version 4 signing will not be used for all requests (false, the default). This attribute maps to the `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` property in the AWS SDK for .NET.

The parent of the `<s3>` element is the `<aws>` element.

The `<s3>` element contains no child elements.

The following is an example of the `<s3>` element in use:

```
<s3 useSignatureVersion4="true" />
```

Amazon Web Services Asynchronous APIs for .NET

Asynchronous API for .NET Framework 4.5, Windows Store, and Windows Phone 8

The AWS SDK for .NET uses the new task-based asynchronous pattern for .NET Framework version 4.5, Windows Store, and Windows Phone 8. You can use the `async` and `await` keywords to perform and manage asynchronous operations for all AWS products without blocking.

To learn more about the task-based asynchronous pattern, see Task-based Asynchronous Pattern (TAP) on MSDN.

Asynchronous API for .NET Framework 3.5

The AWS SDK for .NET supports asynchronous (async) versions of most of the method calls exposed by the .NET client classes. The async methods enable you to call AWS services without having your code block on the response from the service. For example, you can make a request to write data to Amazon S3 or DynamoDB and then have your code continue to do other work while AWS processes the requests.

Syntax of Async Request Methods

There are two phases to making an asynchronous request to an AWS service. The first is to call the `Begin` method for the request. This method initiates the asynchronous operation. The corresponding `End` method retrieves the response from the service and also provides an opportunity to handle exceptions that might have occurred during the operation.

Note

Calling the `End` method is not required. Assuming no errors are encountered, the asynchronous operation will complete whether or not you call `End`.

Begin Method Syntax

In addition to taking a request object parameter, such as PutItemRequest, the async `Begin` methods take two additional parameters: a callback function and a state object. Instead of returning a service response object, the `Begin` methods return a result of type `IAsyncResult`. For the definition of this type, go to the MSDN documentation.

Synchronous Method

```
PutItemResponse PutItem(
  PutItemRequest putItemRequest
)
```

Asynchronous Method

```
IAsyncResult BeginPutItem(
  GetSessionTokenRequest getSessionTokenRequest, {AsyncCallback callback}, {Object state}
)
```

AsyncCallback Callback

The callback function is called when the asynchronous operation is complete. When the function is called, it receives a single parameter of type IAsyncResult. The callback function has the following signature.

```
void Callback(IAsyncResult asyncResult)
```

Object State

The third parameter, `state`, is a user-defined object that is made available to the callback function as the `AsyncState` property of the `asyncResult` parameter, that is, `asyncResult.AsyncState`.

Calling Patterns

- Passing a callback function and a state object.
- Passing a callback function, but passing null for the state object.
- Passing null for both the callback function and the state object.

This topic provides an example of each of these patterns.

Using IAsyncResult.AsyncWaitHandle

In some circumstances, the code that calls the `Begin` method might need to enable another method that it calls to wait on the completion of the asynchronous operation. In these situations, it can pass the method the `WaitHandle` returned by the `IAsyncResult.AsyncWaitHandle` property of the `IAsyncResult` return value. The method can then wait for the asynchronous operation to complete by calling `WaitOne` on this `WaitHandle`.

Examples

All of the following examples assume the following initialization code.

```
public static void TestPutObjectAsync()
{
   // Create a client AmazonS3Client
   client = new AmazonS3Client();

   PutObjectResponse response;
   IAsyncResult asyncResult;

   //
   // Create a PutObject request
   //
   // You will need to use your own bucket name below in order
   // to run this sample code.
   //
   PutObjectRequest request = new PutObjectRequest
   {
     BucketName = "{PUT YOUR OWN EXISTING BUCKET NAME HERE}",
     Key = "Item",
     ContentBody = "This is sample content..."
   };

   //
   // additional example code
   //
}
```

No Callback Specified

The following example code calls `BeginPutObject`, performs some work, and then calls `EndPutObject` to retrieve the service response. The call to `EndPutObject` is enclosed in a `try` block to catch any exceptions that might have been thrown during the operation.

```
asyncResult = client.BeginPutObject(request, null, null);
while ( ! asyncResult.IsCompleted ) {
   //
   // Do some work here
   //
}
try {
   response = client.EndPutObject(asyncResult);
}
catch (AmazonS3Exception s3Exception) {
   //
```

```
    // Code to process exception
    //
  }
```

Simple Callback

This example assumes the following callback function has been defined.

```
public static void SimpleCallback(IAsyncResult asyncResult)
{
  Console.WriteLine("Finished PutObject operation with simple callback");
}
```

The following line of code calls `BeginPutObject` and specifies the above callback function. When the `PutObject` operation is complete, the callback function is called. The call to `BeginPutObject` specifies `null` for the `state` parameter because the simple callback function does not access the `AsyncState` property of the `asyncResult` parameter. Neither the calling code or the callback function call `EndPutObject`. Therefore, the service response is effectively discarded and any exceptions that occur during the operation are ignored.

```
asyncResult = client.BeginPutObject(request, SimpleCallback, null);
```

Callback with Client

This example assumes the following callback function has been defined.

```
public static void CallbackWithClient(IAsyncResult asyncResult)
{
  try {
    AmazonS3Client s3Client = (AmazonS3Client) asyncResult.AsyncState;
    PutObjectResponse response = s3Client.EndPutObject(asyncResult);
    Console.WriteLine("Finished PutObject operation with client callback");
  }
  catch (AmazonS3Exception s3Exception) {
    //
    // Code to process exception
    //
  }
}
```

The following line of code calls `BeginPutObject` and specifies the preceding callback function. When the `PutObject` operation is complete, the callback function is called. In this example, the call to `BeginPutObject` specifies the Amazon S3 client object for the `state` parameter. The callback function uses the client to call the `EndPutObject` method to retrieve the server response. Because any exceptions that occurred during the operation will be received when the callback calls `EndPutObject`, this call is placed within a `try` block.

```
asyncResult = client.BeginPutObject(request, CallbackWithClient, client);
```

Callback with State Object

This example assumes the following class and callback function have been defined.

```
class ClientState
{
  AmazonS3Client client;
  DateTime startTime;

  public AmazonS3Client Client
  {
    get { return client; }
    set { client = value; }
  }

  public DateTime Start
  {
    get { return startTime; }
    set { startTime = value; }
  }
}
```

```
public static void CallbackWithState(IAsyncResult asyncResult)
{
  try {
    ClientState state = asyncResult.AsyncState as ClientState;
    AmazonS3Client s3Client = (AmazonS3Client)state.Client;
    PutObjectResponse response = state.Client.EndPutObject(asyncResult);
    Console.WriteLine("Finished PutObject. Elapsed time: {0}",
      (DateTime.Now - state.Start).ToString());
  }
  catch (AmazonS3Exception s3Exception) {
    //
    // Code to process exception
    //
  }
}
```

The following line of code calls `BeginPutObject` and specifies the above callback function. When the `PutObject` operation is complete, the callback function is called. In this example, the call to `BeginPutObject` specifies, for the `state` parameter, an instance of the `ClientState` class defined previously. This class embeds the Amazon S3 client as well as the time at which `BeginPutObject` is called. The callback function uses the Amazon S3 client object to call the `EndPutObject` method to retrieve the server response. The callback also extracts the start time for the operation and uses it to print the time it took for the asynchronous operation to complete.

As in the previous examples, because exceptions that occur during the operation are received when `EndPutObject` is called, this call is placed within a `try` block.

```
asyncResult = client.BeginPutObject(
  request, CallbackWithState, new ClientState { Client = client, Start = DateTime.Now } );
```

Complete Sample

The following code sample demonstrates the patterns you can use when calling the asynchronous request methods.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading;

using Amazon;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace async_aws_net
{
   class ClientState
   {
      AmazonS3Client client;
      DateTime startTime;

      public AmazonS3Client Client
      {
         get { return client; }
         set { client = value; }
      }

      public DateTime Start
      {
         get { return startTime; }
         set { startTime = value; }
      }
   }

   class Program
   {
      public static void Main(string[] args)
      {
         TestPutObjectAsync();
      }

      public static void SimpleCallback(IAsyncResult asyncResult)
      {
        Console.WriteLine("Finished PutObject operation with simple callback");
        Console.Write("\n\n");
    }

    public static void CallbackWithClient(IAsyncResult asyncResult)
    {
        try {
            AmazonS3Client s3Client = (AmazonS3Client) asyncResult.AsyncState;
            PutObjectResponse response = s3Client.EndPutObject(asyncResult);
            Console.WriteLine("Finished PutObject operation with client callback");
            Console.WriteLine("Service Response:");
            Console.WriteLine("-----------------");
            Console.WriteLine(response);
            Console.Write("\n\n");
        }
        catch (AmazonS3Exception s3Exception) {
            //
            // Code to process exception
            //
        }
    }

    public static void CallbackWithState(IAsyncResult asyncResult)
    {
        try {
            ClientState state = asyncResult.AsyncState as ClientState;
            AmazonS3Client s3Client = (AmazonS3Client)state.Client;
            PutObjectResponse response = state.Client.EndPutObject(asyncResult);
            Console.WriteLine(
                "Finished PutObject operation with state callback that started at {0}",
                (DateTime.Now - state.Start).ToString() + state.Start);
            Console.WriteLine("Service Response:");
            Console.WriteLine("-----------------");
            Console.WriteLine(response);
            Console.Write("\n\n");
        }
        catch (AmazonS3Exception s3Exception) {
            //
            // Code to process exception
            //
        }
    }

    public static void TestPutObjectAsync()
    {
        // Create a client
        AmazonS3Client client = new AmazonS3Client();

        PutObjectResponse response;
        IAsyncResult asyncResult;
```

```
//
// Create a PutObject request
//
// You will need to change the BucketName below in order to run this
// sample code.
//
PutObjectRequest request = new PutObjectRequest
{
  BucketName = "PUT-YOUR-OWN-EXISTING-BUCKET-NAME-HERE",
  Key = "Item",
  ContentBody = "This is sample content..."
};

response = client.PutObject(request);
Console.WriteLine("Finished PutObject operation for {0}.", request.Key);
Console.WriteLine("Service Response:");
Console.WriteLine("-----------------");
Console.WriteLine("{0}", response);
Console.Write("\n\n");

request.Key = "Item1";
asyncResult = client.BeginPutObject(request, null, null);
while ( ! asyncResult.IsCompleted ) {
  //
  // Do some work here
  //
}
try {
  response = client.EndPutObject(asyncResult);
}
catch (AmazonS3Exception s3Exception) {
  //
  // Code to process exception
  //
}

Console.WriteLine("Finished Async PutObject operation for {0}.", request.Key );
Console.WriteLine("Service Response:");
Console.WriteLine("-----------------");
Console.WriteLine(response);
Console.Write("\n\n");

request.Key = "Item2";
asyncResult = client.BeginPutObject(request, SimpleCallback, null);

request.Key = "Item3";
asyncResult = client.BeginPutObject(request, CallbackWithClient, client);

request.Key = "Item4";
```

```
            asyncResult = client.BeginPutObject(request, CallbackWithState,
               new ClientState { Client = client, Start = DateTime.Now } );

            Thread.Sleep( TimeSpan.FromSeconds(5) );
         }
      }
   }
```

See Also

- Getting Started with the AWS SDK for .NET
- Programming with the AWS SDK for .NET

Retries and Timeouts

The AWS SDK for .NET allows you to configure the number of retries and the timeout values for HTTP requests to AWS services. If the default values for retries and timeouts are not appropriate for your application, you can adjust them for your specific requirements, but it is important to understand how doing so will affect the behavior of your application.

To determine which values to use for retries and timeouts, consider the following:

- How should the AWS SDK for .NET and your application respond when network connectivity degrades or an AWS service is unreachable? Do you want the call to fail fast, or is it appropriate for the call to keep retrying on your behalf?
- Is your application a user-facing application or website that must be responsive, or is it a background processing job that has more tolerance for increased latencies?
- Is the application deployed on a reliable network with low latency, or it is deployed at a remote location with unreliable connectivity?

Retries

The AWS SDK for .NET will retry requests that fail due to server-side throttling or dropped connections. You can use the `MaxErrorRetry` property of the ClientConfig class to specify the number of retries at the service client level. the AWS SDK for .NET will retry the operation the specified number of times before failing and throwing an exception. By default, the `MaxErrorRetry` property is set to 4, except for the AmazonDynamoDBConfig class, which defaults to 10 retries. When a retry occurs, it increases the latency of your request. You should configure your retries based on your application limits for total request latency and error rates.

Timeouts

The AWS SDK for .NET allows you to configure the request timeout and socket read/write timeout values at the service client level. These values are specified in the `Timeout` and the `ReadWriteTimeout` properties of the ClientConfig class, respectively. These values are passed on as the `Timeout` and `ReadWriteTimeout` properties of the HttpWebRequest objects created by the AWS service client object. By default, the `Timeout` value is 100 seconds and the `ReadWriteTimeout` value is 300 seconds.

When your network has high latency, or conditions exist that cause an operation to be retried, using long timeout values and a high number of retries can cause some SDK operations to seem unresponsive.

Note

The version of the AWS SDK for .NET that targets the portable class library (PCL) uses the HttpClient class instead of the `HttpWebRequest` class, and supports the Timeout property only.

The following are the exceptions to the default timeout values. These values are overridden when you explicitly set the timeout values.

- `Timeout` and `ReadWriteTimeout` are set to the maximum values if the method being called uploads a stream, such as AmazonS3Client.PutObject(), AmazonS3Client.UploadPart(), AmazonGlacierClient.UploadArchive(), and so on.
- The version of the AWS SDK for .NET that targets the .NET Framework 4.5 sets `Timeout` and `ReadWriteTimeout` to the maximum values for all AmazonS3Client and AmazonGlacierClient objects.
- The version of the AWS SDK for .NET that targets the portable class library (PCL) sets `Timeout` to the maximum value for all AmazonS3Client and AmazonGlacierClient objects.

Example

The following example shows how to specify a maximum of 2 retries, a timeout of 10 seconds, and a read/write timeout of 10 seconds for an AmazonS3Client object.

```
var client =  new AmazonS3Client(
  new AmazonS3Config
  {
    Timeout = TimeSpan.FromSeconds(10),            // Default value is 100 seconds
    ReadWriteTimeout = TimeSpan.FromSeconds(10),   // Default value is 300 seconds
    MaxErrorRetry = 2                              // Default value is 4 retries
  });
```

AWS SDK for .NET Examples

The following examples demonstrate how to use the AWS SDK for .NET to work with individual AWS services.

Additional samples are available on GitHub.

Before you begin, be sure you have set up the AWS SDK for .NET and review Programming with the AWS SDK for .NET.

AWS CloudFormation Examples

The AWS SDK for .NET supports AWS CloudFormation, which creates and provisions AWS infrastructure deployments predictably and repeatedly. For more information, see *CloudFormation Getting Started Guide*.

The following example shows how to use the low-level APIs to list accessible resources in CloudFormation.

```
// using Amazon.CloudFormation;
// using Amazon.CloudFormation.Model;

var client = new AmazonCloudFormationClient();
var request = new DescribeStacksRequest();
var response = client.DescribeStacks(request);

foreach (var stack in response.Stacks)
{
  Console.WriteLine("Stack: {0}", stack.StackName);
  Console.WriteLine("  Status: {0}", stack.StackStatus);
  Console.WriteLine("  Created: {0}", stack.CreationTime);

  var ps = stack.Parameters;

  if (ps.Any())
  {
    Console.WriteLine("  Parameters:");

    foreach (var p in ps)
    {
      Console.WriteLine("    {0} = {1}",
        p.ParameterKey, p.ParameterValue);
    }

  }

}
```

For related API reference information, see Amazon.CloudFormation and Amazon.CloudFormation.Model in the *AWS SDK for .NET API Reference*.

Amazon DynamoDB Examples

The AWS SDK for .NET supports Amazon DynamoDB, which is a fast NoSQL database service offered by AWS. The SDK provides three programming models for communicating with DynamoDB: the *low-level* model, the *document* model, and the *object persistence* model.

The following information introduces these models and their APIs, provides examples for how and when to use them, and gives you links to additional DynamoDB programming resources in the AWS SDK for .NET.

Using Expressions with Amazon DynamoDB and the AWS SDK for .NET

The following code examples demonstrate how to use the AWS SDK for .NET to program DynamoDB with expressions. *Expressions* denote the attributes you want to read from an item in a DynamoDB table. You also use expressions when writing an item, to indicate any conditions that must be met (also known as a *conditional update*) and how the attributes are to be updated. Some update examples are replacing the attribute with a new value, or adding new data to a list or a map. For more information, see Reading and Writing Items Using Expressions.

Sample Data

The code examples in this topic rely on the following two example items in a DynamoDB table named `ProductCatalog`. These items describe information about product entries in a fictitious bicycle store catalog. These items are based on the example provided in Case Study: A ProductCatalog Item. The data type descriptors such as `BOOL`, `L`, `M`, `N`, `NS`, `S`, and `SS` correspond to those in the JSON Data Format.

```
{
  "Id": {
    "N": "205"
  },
  "Title": {
    "S": "20-Bicycle 205"
  },
  "Description": {
    "S": "205 description"
  },
  "BicycleType": {
    "S": "Hybrid"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "500"
  },
  "Gender": {
    "S": "B"
  },
  "Color": {
    "SS": [
      "Red",
      "Black"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
            "S": "http://example/products/205_rear.jpg"
          }
        }
```

```
          },
          {
            "M": {
              "SideView": {
                "S": "http://example/products/205_left_side.jpg"
              }
            }
          }
        ]
      },
      "ProductReviews": {
        "M": {
          "FiveStar": {
            "SS": [
              "Excellent! Can't recommend it highly enough! Buy it!",
              "Do yourself a favor and buy this."
            ]
          },
          "OneStar": {
            "SS": [
              "Terrible product! Do not buy this."
            ]
          }
        }
      }
    },
    {
      "Id": {
        "N": "301"
      },
      "Title": {
        "S": "18-Bicycle 301"
      },
      "Description": {
        "S": "301 description"
      },
      "BicycleType": {
        "S": "Road"
      },
      "Brand": {
        "S": "Brand-Company C"
      },
      "Price": {
        "N": "185"
      },
      "Gender": {
        "S": "F"
      },
      "Color": {
        "SS": [
          "Blue",
          "Silver"
        ]
      },
      "ProductCategory": {
        "S": "Bike"
      },
      "InStock": {
        "BOOL": true
      },
      "QuantityOnHand": {
        "N": "3"
      },
      "RelatedItems": {
        "NS": [
          "801",
          "822",
          "979"
        ]
      },
      "Pictures": {
        "L": [
          {
            "M": {
              "FrontView": {
                "S": "http://example/products/301_front.jpg"
              }
            }
          },
          {
            "M": {
              "RearView": {
                "S": "http://example/products/301_rear.jpg"
              }
            }
          },
          {
            "M": {
              "SideView": {
                "S": "http://example/products/301_left_side.jpg"
              }
            }
          }
        ]
      },
      "ProductReviews": {
        "M": {
          "FiveStar": {
            "SS": [
              "My daughter really enjoyed this bike!"
            ]
          },
          "ThreeStar": {
            "SS": [
              "This bike was okay, but I would have preferred it in my color.",
                  "Fun to ride."
            ]
          }
        }
      }
    }
```

Get a Single Item by Using Expressions and the Item's Primary Key

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` method and a set of expressions to get and then print the item that has an `Id` of `205`. Only the following attributes of the item are returned: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures`, and `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
  },
  Key = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "205" } }
  },
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#ri` to represent the `RelatedItems` attribute. The call to `PrintItem` refers to a custom function as described in Print an Item.

Get Multiple Items by Using Expressions and the Table's Primary Key

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` method and a set of expressions to get and then print the item that has an `Id` of `301`, but only if the value of `Price` is greater than `150`. Only the following attributes of the item are returned: `Id`, `Title`, and all of the `ThreeStar` attributes in `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new QueryRequest
{
  TableName = "ProductCatalog",
  KeyConditions = new Dictionary<string,Condition>
  {
    { "Id", new Condition()
      {
        ComparisonOperator = ComparisonOperator.EQ,
        AttributeValueList = new List<AttributeValue>
        {
          new AttributeValue { N = "301" }
        }
      }
    }
  },
  ProjectionExpression = "Id, Title, #pr.ThreeStar",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#p", "Price" }
  },
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":val", new AttributeValue { N = "150" } }
  },
  FilterExpression = "#p > :val"
};
var response = client.Query(request);

foreach (var item in response.Items)
{
  // Write out the first page of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#p` to represent the `Price` attribute. `#pr.ThreeStar` specifies to return only the `ThreeStar` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:val` to represent the value `150`. The `FilterExpression` property specifies that `#p` (`Price`) must be greater than `:val` (`150`). The call to `PrintItem` refers to a custom function as described in Print an Item.

Get Multiple Items by Using Expressions and Other Item Attributes

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` method and a set of expressions to get and then print all items that have a `ProductCategory` of `Bike`. Only the following attributes of the item are returned: `Id`, `Title`, and all of the attributes in `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new ScanRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, #pr",
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":catg", new AttributeValue { S = "Bike" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#pc", "ProductCategory" }
  },
  FilterExpression = "#pc = :catg",
};
var response = client.Scan(request);

foreach (var item in response.Items)
{
  // Write out the first page/scan of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#pc` to represent the `ProductCategory` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:catg` to represent the value `Bike`. The `FilterExpression` property specifies that `#pc` (`ProductCategory`) must be equal to `:catg` (`Bike`). The call to `PrintItem` refers to a custom function as described in Print an Item.

Print an Item

The following example shows how to print an item's attributes and values. This example is used in the preceding examples that show how to Get a Single Item by Using Expressions and the Item's Primary Key, Get Multiple Items by Using Expressions and the Table's Primary Key, and Get Multiple Items by Using Expressions and Other Item Attributes.

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
  foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
  {
    Console.Write(kvp.Key + " = ");
    PrintValue(kvp.Value);
  }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
  // Binary attribute value.
  if (value.B != null)
  {
    Console.Write("Binary data");
  }
  // Binary set attribute value.
  else if (value.BS.Count > 0)
  {
    foreach (var bValue in value.BS)
    {
      Console.Write("\n  Binary data");
    }
  }
  // List attribute value.
  else if (value.L.Count > 0)
  {
    foreach (AttributeValue attr in value.L)
    {
      PrintValue(attr);
    }
  }
  // Map attribute value.
  else if (value.M.Count > 0)
  {
    Console.Write("\n");
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
  }

  Console.Write("\n");
}
```

In the preceding example, each attribute value has several data-type-specific properties that can be evaluated to determine the correct format to print the attribute. These properties include `B`, `BOOL`, `BS`, `L`, `M`, `N`, `NS`, `NULL`, `S`, and `SS`, which correspond to those in the JSON Data Format. For properties such as `B`, `N`, `NULL`, and `S`, if the corresponding property is not `null`, then the attribute is of the corresponding non-`null` data type. For properties such as `BS`, `L`, `M`, `NS`, and `SS`, if `Count` is greater than zero, then the attribute is of the corresponding non-zero-value data type. If all of the attribute's data-type-specific properties are either `null` or the `Count` equals zero, then the attribute corresponds to the `BOOL` data type.

Create or Replace an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` method and a set of expressions to update the item that has a `Title` of `18-Bicycle 301`. If the item doesn't already exist, a new item is added.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new PutItemRequest
{
  TableName = "ProductCatalog",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product",
  // CreateItemData() is a custom function.
  Item = CreateItemData()
};
client.PutItem(request);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:product` to represent the value `18-Bicycle 301`. The `ConditionExpression` property specifies that `#title` (`Title`) must be equal to `:product` (`18-Bicycle 301`). The call to `CreateItemData` refers to the following custom function:

```
// using Amazon.DynamoDBv2.Model;

// Provides a sample item that can be added to a table.
public static Dictionary<string, AttributeValue> CreateItemData()
{
  var itemData = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "301" } },
    { "Title", new AttributeValue { S = "18\" Girl's Bike" } },
    { "BicycleType", new AttributeValue { S = "Road" } },
    { "Brand" , new AttributeValue { S = "Brand-Company C" } },
    { "Color", new AttributeValue { SS = new List<string>{ "Blue", "Silver" } } },
    { "Description", new AttributeValue { S = "301 description" } },
    { "Gender", new AttributeValue { S = "F" } },
    { "InStock", new AttributeValue { BOOL = true } },
    { "Pictures", new AttributeValue { L = new List<AttributeValue>{
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "FrontView", new AttributeValue { S = "http://example/products/301_front.jpg" }
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "RearView", new AttributeValue {S = "http://example/products/301_rear.jpg" } } }
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "SideView", new AttributeValue { S = "http://example/products/301_left_side.jpg"
    } } },
    { "Price", new AttributeValue { N = "185" } },
    { "ProductCategory", new AttributeValue { S = "Bike" } },
    { "ProductReviews", new AttributeValue { M = new Dictionary<string,AttributeValue>{
      { "FiveStar", new AttributeValue { SS = new List<string>{
        "My daughter really enjoyed this bike!" } } },
      { "OneStar", new AttributeValue { SS = new List<string>{
        "Fun to ride.",
        "This bike was okay, but I would have preferred it in my color." } } }
    } } },
    { "QuantityOnHand", new AttributeValue { N = "3" } },
    { "RelatedItems", new AttributeValue { NS = new List<string>{ "979", "822", "801" } } }
  };

  return itemData;
}
```

In the preceding example, an example item with sample data is returned to the caller. A series of attributes and corresponding values are constructed, using data types such as `BOOL`, `L`, `M`, `N`, `NS`, `S`, and `SS`, which correspond to those in the JSON Data Format.

Update an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` method and a set of expressions to change the `Title` to `18" Girl's Bike` for the item with `Id` of `301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
  },
  UpdateExpression = "SET #title = :newproduct"
};
client.UpdateItem(request);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:newproduct` to represent the value `18" Girl's Bike`. The `UpdateExpression` property specifies to change `#title` (`Title`) to `:newproduct` (`18" Girl's Bike`).

Delete an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` method and a set of expressions to delete the item with `Id` of `301`, but only if the item's `Title` is `18-Bicycle 301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:product` to represent the value `18-Bicycle 301`. The `ConditionExpression` property specifies that `#title` (`Title`) must equal `:product` (`18-Bicycle 301`).

More Info

For more information and code examples, see:

- DynamoDB Series - Expressions
- Accessing Item Attributes with Projection Expressions
- Using Placeholders for Attribute Names and Values
- Specifying Conditions with Condition Expressions
- Modifying Items and Attributes with Update Expressions
- Working with Items Using the AWS SDK for .NET Low-Level API
- Querying Tables Using the AWS SDK for .NET Low-Level API
- Scanning Tables Using the AWS SDK for .NET Low-Level API
- Working with Local Secondary Indexes Using the AWS SDK for .NET Low-Level API
- Working with Global Secondary Indexes Using the AWS SDK for .NET Low-Level API

JSON Support in Amazon DynamoDB with the AWS SDK for .NET

The AWS SDK for .NET supports JSON data when working with Amazon DynamoDB. This enables you to more easily get JSON-formatted data from, and insert JSON documents into, DynamoDB tables.

Get Data from a DynamoDB Table in JSON Format

The following example shows how to get data from a DynamoDB table in JSON format:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
//   {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
//   {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
//   }
```

In the preceding example, the `ToJson` method of the `Document` class converts an item from the table into a JSON-formatted string. The item is retrieved through the `GetItem` method of the `Table` class. To determine the item to get, in this example, the `GetItem` method uses the hash-and-range primary key of the target item. To determine the table to get the item from, the `LoadTable` method of the `Table` class uses an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

Insert JSON Format Data into a DynamoDB Table

The following example shows how to use JSON format to insert an item into a DynamoDB table:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var jsonText = "{\"Id\":6,\"Type\":\"Bird\",\"Name\":\"Tweety\"}";
var item = Document.FromJson(jsonText);

table.PutItem(item);
```

In the preceding example, the `FromJson` method of the `Document` class converts a JSON-formatted string into an item. The item is inserted into the table through the `PutItem` method of the `Table` class, which uses the instance of the `Document` class that contains the item. To determine the table to insert the item into, the `LoadTable` method of the `Table` class is called, specifying an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

DynamoDB Data Type Conversions to JSON

Whenever you call the `ToJson` method of the `Document` class, and then on the resulting JSON data you call the `FromJson` method to convert the JSON data back into an instance of a `Document` class, some DynamoDB data types will not be converted as expected. Specifically:

- DynamoDB sets (the `SS`, `NS`, and `BS` types) will be converted to JSON arrays.
- DynamoDB binary scalars and sets (the `B` and `BS` types) will be converted to base64-encoded JSON strings or lists of strings.

  In this scenario, you must call the `DecodeBase64Attributes` method of the `Document` class to replace the base64-encoded JSON data with the correct binary representation. The following example replaces a base64-encoded binary scalar item attribute in an instance of a `Document` class, named `Picture`, with the correct binary representation. This example also does the same for a base64-encoded binary set item attribute in the same instance of the `Document` class, named `RelatedPictures`:

  ```
  item.DecodeBase64Attributes("Picture", "RelatedPictures");
  ```

More Info

For more information and examples of programming JSON with DynamoDB with the AWS SDK for .NET, see:

- DynamoDB JSON Support
- Amazon DynamoDB Update - JSON, Expanded Free Tier, Flexible Scaling, Larger Items

Managing ASP.NET Session State with Amazon DynamoDB

ASP.NET applications often store session state data in memory. However, this approach doesn't scale well. After the application grows beyond a single web server, the session state must be shared between servers. A common solution is to set up a dedicated session-state server with Microsoft SQL Server, but this approach also has drawbacks: you must administer another machine; the session-state server is a single point of failure; and the session-state server itself can become a performance bottleneck.

DynamoDB, a NoSQL database store from Amazon Web Services (AWS), provides an effective solution for sharing session state across web servers without incurring any of these drawbacks.

> **Note**
>
> Regardless of the solution you choose, be aware that Amazon DynamoDB enforces limits on the size of an item. None of the records you store in DynamoDB can exceed this limit. For more information, see Limits in DynamoDB in the *DynamoDB Developer Guide*.

The AWS SDK for .NET includes `AWS.SessionProvider.dll`, which contains an ASP.NET session state provider. It also includes the *AmazonDynamoDBSessionProviderSample* sample, which demonstrates how to use Amazon DynamoDB as a session state provider.

For more information about using session state with ASP.NET applications, go to the MSDN documentation.

Create the ASP.NET_SessionState Table

When your application starts, it looks for an Amazon DynamoDB table named, by default, `ASP.NET_SessionState`. We recommend you create this table before you run your application for the first time.

To create the ASP.NET_SessionState table

1. Choose **Create Table**. The **Create Table** wizard opens.
2. In the **Table name** text box, enter `ASP.NET_SessionState`.
3. In the **Primary key** field, enter `SessionId` and set the type to `String`.
4. When all your options are entered as you want them, choose **Create**.

The ASP.NET_SessionState table is ready for use when its status changes from `CREATING` to `ACTIVE`.

> **Note**
>
> If you decide not to create the table beforehand, the session state provider will create the table during its initialization. See the web.config options below for a list of attributes that act as configuration parameters for the session state table. If the provider creates the table, it will use these parameters.

Configure the Session State Provider

To configure an ASP.NET application to use DynamoDB as the session-state server

1. Add references to both `AWSSDK.dll` and `AWS.SessionProvider.dll` to your Visual Studio ASP.NET project. These assemblies are available by installing the AWS SDK for .NET. You can also install them by using NuGet.

   In earlier versions of the SDK, the functionality for the session state provider was contained in `AWS.Extension.dll`. To improve usability, the functionality was moved to `AWS.SessionProvider.dll`. For more information, see the blog post AWS.Extension Renaming.

2. Edit your application's Web.config file. In the `system.web` element, replace the existing `sessionState` element with the following XML fragment:

```xml
<sessionState timeout="20" mode="Custom" customProvider="DynamoDBSessionStoreProvider">
  <providers>
    <add name="DynamoDBSessionStoreProvider"
         type="Amazon.SessionProvider.DynamoDBSessionStateStore"
         AWSProfileName="{profile_name}"
         Region="us-west-2" />
  </providers>
</sessionState>
```

   The profile represents the AWS credentials that are used to communicate with DynamoDB to store and retrieve the session state. If you are using the AWS SDK for .NET and are specifying a profile in the `appSettings` section of your application's Web.config file, you do not need to specify a profile in the `providers` section; the AWS .NET client code will discover it at run time. For more information, see Configuring Your AWS SDK for .NET Application.

   If the web server is running on an Amazon EC2 instance configured to use IAM roles for EC2 instances, then you do not need to specify any credentials in the Web.config file. In this case, the AWS .NET client will use the IAM role credentials. For more information, see Granting Access Using an IAM Role and Security Considerations.

Web.config Options

You can use the following configuration attributes in the `providers` section of your Web.config file:

AWSAccessKey

Access key ID to use. This can be set either in the `providers` or `appSettings` section. We recommend not using this setting. Instead, specify credentials by using `AWSProfileName` to specify a profile.

AWSSecretKey

Secret key to use. This can be set either in the `providers` or `appSettings` section. We recommend not using this setting. Instead, specify credentials by using `AWSProfileName` to specify a profile.

AWSProfileName

The profile name associated with the credentials you want to use. For more information, see Configuring Your AWS SDK for .NET Application.

Region

Required `string` attribute. The AWS region in which to use Amazon DynamoDB. For a list of AWS regions, see Regions and Endpoints: DynamoDB.

Application

Optional `string` attribute. The value of the `Application` attribute is used to partition the session data in the table so that the table can be used for more than one application.

Table

Optional `string` attribute. The name of the table used to store session data. The default is `ASP.NET_SessionState`.

ReadCapacityUnits

Optional `int` attribute. The read capacity units to use if the provider creates the table. The default is 10.

WriteCapacityUnits

Optional `int` attribute. The write capacity units to use if the provider creates the table. The default is 5.

CreateIfNotExist

Optional `boolean` attribute. The `CreateIfNotExist` attribute controls whether the provider will auto-create the table if it doesn't exist. The default is true. If this flag is set to false and the table doesn't exist, an exception will be thrown.

Security Considerations

After the DynamoDB table is created and the application is configured, sessions can be used as with any other session provider.

As a security best practice, we recommend you run your applications with the credentials of an *IAM User Guide* user. You can use either the IAM Management Console or the AWS Toolkit for Visual Studio to create IAM users and define access policies.

The session state provider needs to be able to call the DeleteItem, DescribeTable, GetItem, PutItem, and UpdateItem operations for the table that stores the session data. The sample policy below can be used to restrict the IAM user to only the operations needed by the provider for an instance of DynamoDB running in us-east-1:

```
{ "Version" : "2012-10-17",
"Statement" : [
  {
    "Sid" : "1",
    "Effect" : "Allow",
    "Action" : [
        "dynamodb:DeleteItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem"
    ],
    "Resource" : "arn:aws:dynamodb:|region_api_default|:{<YOUR-AWS-ACCOUNT-ID>}:table/ASP.NE
    }
  ]
}
```

Low-Level Model

The low-level programming model wraps direct calls to the DynamoDB service. You access this model through the Amazon.DynamoDBv2 namespace.

Of the three models, the low-level model requires you to write the most code. For example, you must convert .NET data types to their equivalents in DynamoDB. However, this model gives you access to the most features.

The following examples show you how to use the low-level model to create a table, modify a table, and insert items into a table in DynamoDB.

Creating a Table

In the following example, you create a table by using the `CreateTable` method of the `AmazonDynamoDBClient` class. The `CreateTable` method uses an instance of the `CreateTableRequest` class that contains characteristics such as required item attribute names, primary key definition, and throughput capacity. The `CreateTable` method returns an instance of the `CreateTableResponse` class.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request = new CreateTableRequest
{
  TableName = "AnimalsInventory",
  AttributeDefinitions = new List<AttributeDefinition>
  {
    new AttributeDefinition
    {
      AttributeName = "Id",
      // "S" = string, "N" = number, and so on.
      AttributeType = "N"
    },
    new AttributeDefinition
    {
      AttributeName = "Type",
      AttributeType = "S"
    }
  },
  KeySchema = new List<KeySchemaElement>
  {
    new KeySchemaElement
    {
      AttributeName = "Id",
      // "HASH" = hash key, "RANGE" = range key.
      KeyType = "HASH"
    },
    new KeySchemaElement
    {
      AttributeName = "Type",
      KeyType = "RANGE"
    },
  },
  ProvisionedThroughput = new ProvisionedThroughput
  {
    ReadCapacityUnits = 10,
    WriteCapacityUnits = 5
  },
};

var response = client.CreateTable(request);
```

```
Console.WriteLine("Table created with request ID: " +
  response.ResponseMetadata.RequestId);
```

Verifying That a Table is Ready to Modify

Before you can change or modify a table, the table has to be ready for modification. The following example shows how to use the low-level model to verify that a table in DynamoDB is ready. In this example, the target table to check is referenced through the `DescribeTable` method of the `AmazonDynamoDBClient` class. Every five seconds, the code checks the value of the table's `TableStatus` property. When the status is set to `ACTIVE`, the table is ready to be modified.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var status = "";

do
{
  // Wait 5 seconds before checking (again).
  System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

  try
  {
    var response = client.DescribeTable(new DescribeTableRequest
    {
      TableName = "AnimalsInventory"
    });

    Console.WriteLine("Table = {0}, Status = {1}",
      response.Table.TableName,
      response.Table.TableStatus);

    status = response.Table.TableStatus;
  }
  catch (ResourceNotFoundException)
  {
    // DescribeTable is eventually consistent. So you might
    //   get resource not found.
  }

} while (status != TableStatus.ACTIVE);
```

Inserting an Item into a Table

In the following example, you use the low-level model to insert two items into a table in DynamoDB. Each item is inserted through the `PutItem` method of the `AmazonDynamoDBClient` class, using an instance of the `PutItemRequest` class. Each of the two instances of the `PutItemRequest` class takes the name of the table that the items will be inserted in, with a series of item attribute values.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "1" }},
    { "Type", new AttributeValue { S = "Dog" }},
    { "Name", new AttributeValue { S = "Fido" }}
  }
};

var request2 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "2" }},
    { "Type", new AttributeValue { S = "Cat" }},
    { "Name", new AttributeValue { S = "Patches" }}
  }
};

client.PutItem(request1);
client.PutItem(request2);
```

Document Model

The document programming model provides an easier way to work with data in DynamoDB. This model is specifically intended for accessing tables and items in tables. You access this model through the Amazon.DynamoDBv2.DocumentModel namespace.

Compared to the low-level programming model, the document model is easier to code against DynamoDB data. For example, you don't have to convert as many .NET data types to their equivalents in DynamoDB. However, this model doesn't provide access to as many features as the low-level programming model. For example, you can use this model to create, retrieve, update, and delete items in tables. However, to create the tables, you must use the low-level model. Compared to the object persistence model, this model requires you to write more code to store, load, and query .NET objects.

The following examples show you how to use the document model to insert items and get items in tables in DynamoDB.

Inserting an Item into a Table

In the following example, an item is inserted into the table through the `PutItem` method of the `Table` class. The `PutItem` method takes an instance of the `Document` class; the `Document` class is simply a collection of initialized attributes. To determine the table to insert the item into, call the `LoadTable` method of the `Table` class, specifying an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = new Document();

item["Id"] = 3;
item["Type"] = "Horse";
item["Name"] = "Shadow";

table.PutItem(item);
```

Getting an Item from a Table

In the following example, the item is retrieved through the `GetItem` method of the `Table` class. To determine the item to get, the `GetItem` method uses the hash-and-range primary key of the target item. To determine the table to get the item from, the `LoadTable` method of the `Table` class uses an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

Console.WriteLine("Id = " + item["Id"]);
Console.WriteLine("Type = " + item["Type"]);
Console.WriteLine("Name = " + item["Name"]);
```

The preceding example implicitly converts the attribute values for `Id`, `Type`, and `Name` to strings for the `WriteLine` method. You can do explicit conversions by using the various `AsType` methods of the `DynamoDBEntry` class. For example, you could explicitly convert the attribute value for `Id` from a `Primitive` data type to an integer through the `AsInt` method:

```
int id = item["Id"].AsInt();
```

Or, you could simply perform an explicit cast here by using `(int)`:

```
int id = (int)item["Id"];
```

Object Persistence Model

The object persistence programming model is specifically designed for storing, loading, and querying .NET objects in DynamoDB. You access this model through the Amazon.DynamoDBv2.DataModel namespace.

Of the three models, the object persistence model is the easiest to code against whenever you are storing, loading, or querying DynamoDB data. For example, you work with DynamoDB data types directly. However, this model provides access only to operations that store, load, and query .NET objects in DynamoDB. For example, you can use this model to create, retrieve, update, and delete items in tables. However, you must first create your tables using the low-level model, and then use this model to map your .NET classes to the tables.

The following examples show you how to define a .NET class that represents an item, use an instance of the .NET class to insert an item, and use an instance of a .NET object to get an item from a table in DynamoDB.

Defining a .NET Class that Represents an Item in a Table

In the following example, the `DynamoDBTable` attribute specifies the table name, while the `DynamoDBHashKey` and `DynamoDBRangeKey` attributes model the table's hash-and-range primary key.

```
// using Amazon.DynamoDBv2.DataModel;

[DynamoDBTable("AnimalsInventory")]
class Item
{
  [DynamoDBHashKey]
  public int Id { get; set; }
  [DynamoDBRangeKey]
  public string Type { get; set; }
  public string Name { get; set; }
}
```

Using an Instance of the .NET Class to Insert an Item into a Table

In this example, the item is inserted through the `Save` method of the `DynamoDBContext` class, which takes an initialized instance of the .NET class that represents the item. (The instance of the `DynamoDBContext` class is initialized with an instance of the `AmazonDynamoDBClient` class.)

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = new Item
{
  Id = 4,
  Type = "Fish",
  Name = "Goldie"
};

context.Save(item);
```

Using an Instance of a .NET Object to Get an Item from a Table

In this example, the item is retrieved through the `Load` method of the `DynamoDBContext` class, which takes a partially initialized instance of the .NET class that represents the hash-and-range primary key of the item to be retrieved. (As shown previously, the instance of the `DynamoDBContext` class is initialized with an instance of the `AmazonDynamoDBClient` class.)

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = context.Load<Item>(4, "Fish");

Console.WriteLine("Id = {0}", item.Id);
Console.WriteLine("Type = {0}", item.Type);
Console.WriteLine("Name = {0}", item.Name);
```

More Info

Using the |sdk-net| to program |DDB| information and examples**

- DynamoDB APIs
- DynamoDB Series Kickoff
- DynamoDB Series - Document Model
- DynamoDB Series - Conversion Schemas
- DynamoDB Series - Object Persistence Model
- DynamoDB Series - Expressions
- Using Expressions with Amazon DynamoDB and the AWS SDK for .NET
- JSON Support in Amazon DynamoDB with the AWS SDK for .NET
- Managing ASP.NET Session State with Amazon DynamoDB

Low-Level model information and examples

- Working with Tables Using the AWS SDK for .NET Low-Level API
- Working with Items Using the AWS SDK for .NET Low-Level API
- Querying Tables Using the AWS SDK for .NET Low-Level API
- Scanning Tables Using the AWS SDK for .NET Low-Level API
- Working with Local Secondary Indexes Using the AWS SDK for .NET Low-Level API
- Working with Global Secondary Indexes Using the AWS SDK for .NET Low-Level API

Document model information and examples

- DynamoDB Data Types
- DynamoDBEntry
- .NET: Document Model

Object persistance model information and examples

- .NET: Object Persistence Model

Amazon EC2 Examples

The AWS SDK for .NET supports Amazon EC2, which is a web service that provides resizable computing capacity—literally, servers in Amazon's data centers—that you use to build and host your software systems. The Amazon EC2 APIs are provided by the AWSSDK.EC2 assembly.

The Amazon EC2 instances examples are intended to help you get started with Amazon EC2.

The Amazon EC2 Spot instances examples show you how to use Spot instances, which enable you to bid on unused Amazon EC2 capacity and run any instances that you acquire for as long as your bid exceeds the current Spot price. Amazon EC2 changes the Spot price periodically based on supply and demand; customers whose bids meet or exceed it gain access to the available Spot instances. For more information, see Spot Instances in the *Amazon EC2 User Guide for Linux Instances* and Spot Instances in the *Amazon EC2 User Guide for Windows Instances*.

Amazon EC2 Instances Examples

You can access the features of Amazon EC2 using the AWS SDK for .NET. For example, you can create, start, and terminate Amazon EC2 instances.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio a set of C# project templates are installed. So the easiest way to start this project is to open Visual Studio, and then choose **File**, **New Project**, **AWS Sample Projects**, **Compute and Networking**, **AWS EC2 Sample**.

Prerequisites

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET.

Examples

Creating an Amazon EC2 Client

Create an Amazon EC2 client to manage your EC2 resources, such as instances and security groups. This client is represented by an AmazonEC2Client object, which you can create as follows.

```
var ec2Client = new AmazonEC2Client();
```

The permissions for the client object are determined by the policy attached to the profile you specified in the App.config file. By default, we use the region specified in App.config. To use a different region, pass the appropriate RegionEndpoint value to the constructor. For more information, see Regions and Endpoints: EC2 in the *Amazon Web Services General Reference*.

Creating a Security Group in Amazon EC2

In Amazon EC2, a security group acts as a virtual firewall that controls the network traffic for one or more EC2 instances. By default, Amazon EC2 associates your instances with a security group that allows no inbound traffic. You can create a security group that allows your EC2 instances to accept certain traffic. For example, if you need to connect to an EC2 Windows instance, you must configure the security group to allow RDP traffic. You can create a security group by using the Amazon EC2 console or the AWS SDK for .NET.

You create a security group for use in either EC2-Classic or EC2-VPC. For more information about EC2-Classic and EC2-VPC, see Supported Platforms in the *Amazon EC2 User Guide for Windows Instances*.

Alternatively, you can create a security group using the Amazon EC2 console. For more information, see Amazon EC2 Security Groups in the *Amazon EC2 User Guide for Windows Instances*.

Enumerate Your Security Groups

You can enumerate your security groups and check whether a security group exists.

Get the complete list of your security groups using DescribeSecurityGroups with no parameters.

The following example enumerates all of the security groups in the region.

```
static void EnumerateSecurityGroups(AmazonEC2Client ec2Client)
{
  var request = new DescribeSecurityGroupsRequest();
  var response = ec2Client.DescribeSecurityGroups(request);
  List<SecurityGroup> mySGs = response.SecurityGroups;
  foreach (SecurityGroup item in mySGs)
  {
    Console.WriteLine("Security group: " + item.GroupId);
    Console.WriteLine("\tGroupId: " + item.GroupId);
    Console.WriteLine("\tGroupName: " + item.GroupName);
    Console.WriteLine("\tVpcId: " + item.VpcId);

    Console.WriteLine();
  }
}
```

Use DescribeSecurityGroups with a filter.

The following example retrieves only the security groups that belong to the specified VPC.

```
static void EnumerateVpcSecurityGroups(AmazonEC2Client ec2Client, string vpcID)
{
  Filter vpcFilter = new Filter
  {
    Name = "vpc-id",
    Values = new List<string>() { vpcID }
  };

  var request = new DescribeSecurityGroupsRequest();
  request.Filters.Add(vpcFilter);
  var response = ec2Client.DescribeSecurityGroups(request);
  List<SecurityGroup> mySGs = response.SecurityGroups;
  foreach (SecurityGroup item in mySGs)
  {
    Console.WriteLine("Security group: " + item.GroupId);
    Console.WriteLine("\tGroupId: " + item.GroupId);
    Console.WriteLine("\tGroupName: " + item.GroupName);
    Console.WriteLine("\tVpcId: " + item.VpcId);

    Console.WriteLine();
  }
}
```

Create a Security Group

If you attempt to create a security group with a name of an existing security group, CreateSecurityGroup will throw an exception. To avoid this, the following examples search for a security group with the specified name, and return the appropriate SecurityGroup object if one is found.

Create and initialize a CreateSecurityGroupRequest object. Assign a name and description to the `GroupName` and `Description` properties, respectively.

The CreateSecurityGroup method returns a CreateSecurityGroupResponse object. You can get the identifier of the new security group from the response and then use DescribeSecurityGroups with the security group identifier to get the SecurityGroup object for the security group.

```
static SecurityGroup CreateEc2SecurityGroup(
  AmazonEC2Client ec2Client,
  string secGroupName)
{
  // See if a security group with the specified name already exists
  Filter nameFilter = new Filter();
  nameFilter.Name = "group-name";
  nameFilter.Values= new List<string>() { secGroupName };

  var describeRequest = new DescribeSecurityGroupsRequest();
  describeRequest.Filters.Add(nameFilter);
  var describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);

  // If a match was found, return the SecurityGroup object for the security group
  if(describeResponse.SecurityGroups.Count > 0)
  {
    return describeResponse.SecurityGroups[0];
  }

  // Create the security group
  var createRequest = new CreateSecurityGroupRequest();
  createRequest.GroupName = secGroupName;
  createRequest.Description = "My sample security group for EC2-Classic";

  var createResponse = ec2Client.CreateSecurityGroup(createRequest);

  var Groups = new List<string>() { createResponse.GroupId };
  describeRequest = new DescribeSecurityGroupsRequest() { GroupIds = Groups };
  describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);
  return describeResponse.SecurityGroups[0];
}
```

Create and initialize a CreateSecurityGroupRequest object. Assign values to the `GroupName`, `Description`, and `VpcId` properties.

The CreateSecurityGroup method returns a CreateSecurityGroupResponse object. You can get the identifier of the new security group from the response and then use DescribeSecurityGroups with the security group identifier to get the SecurityGroup object for the security group.

```
static SecurityGroup CreateVpcSecurityGroup(
  AmazonEC2Client ec2Client,
  string vpcId,
  string secGroupName)
{
  // See if a security group with the specified name already exists
  Filter nameFilter = new Filter();
  nameFilter.Name = "group-name";
  nameFilter.Values = new List<string>() { secGroupName };

  var describeRequest = new DescribeSecurityGroupsRequest();
  describeRequest.Filters.Add(nameFilter);
  var describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);

  // If a match was found, return the SecurityGroup object for the security group
  if (describeResponse.SecurityGroups.Count > 0)
  {
    return describeResponse.SecurityGroups[0];
  }

  // Create the security group
  var createRequest = new CreateSecurityGroupRequest();
  createRequest.GroupName = secGroupName;
  createRequest.Description = "My sample security group for EC2-VPC";
  createRequest.VpcId = vpcId;

  var createResponse = ec2Client.CreateSecurityGroup(createRequest);

  var Groups = new List<string>() { createResponse.GroupId };
  describeRequest = new DescribeSecurityGroupsRequest() { GroupIds = Groups };
  describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);
  return describeResponse.SecurityGroups[0];
}
```

Add Rules to Your Security Group

Use the following procedure to add a rule to allow inbound traffic on TCP port 3389 (RDP). This enables you to connect to a Windows instance. If you're launching a Linux instance, use TCP port 22 (SSH) instead.

Tip

You can use a service to get the public IP address of your local computer. For example, we provide the following service: http://checkip.amazonaws.com/. To locate another service that provides your IP address, use the search phrase "what is my IP address". If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

The examples in this section follow from the examples in the previous sections. They assume `secGroup` is an existing security group.

1. Create and initialize an IpPermission object.

   ```
   string ipRange = "1.1.1.1/1";
   List<string> ranges = new List<string>() { ipRange };

   var ipPermission = new IpPermission();
   ipPermission.IpProtocol = "tcp";
   ipPermission.FromPort = 3389;
   ipPermission.ToPort = 3389;
   ipPermission.IpRanges = ranges;
   ```

   `IpProtocol`

   The IP protocol.

   `FromPort` **and** `ToPort`

   The beginning and end of the port range. This example specifies a single port, 3389, which is used to communicate with Windows over RDP.

   `IpRanges`

   The IP addresses or address ranges, in CIDR notation. For convenience, this example uses `72.21.198.64/24`, which authorizes network traffic for a single IP address. You can use http://checkip.amazonaws.com/ to determine your own IP addcress.

2. Create and initialize an AuthorizeSecurityGroupIngressRequest object.

   ```
   var ingressRequest = new AuthorizeSecurityGroupIngressRequest();
   ingressRequest.GroupId = secGroup.GroupId;
   ingressRequest.IpPermissions.Add(ipPermission);
   ```

   `GroupId`

   The identifier of the security group.

   `IpPermissions`

   The `IpPermission` object from step 1.

3. (Optional) You can add additional rules to the `IpPermissions` collection before going to the next step.

4. Pass the AuthorizeSecurityGroupIngressRequest object to the AuthorizeSecurityGroupIngress method, which returns an AuthorizeSecurityGroupIngressResponse object. If a matching rule already exists, an AmazonEC2Exception is thrown.

   ```
   try
   {
   ```

```
  var ingressResponse = ec2Client.AuthorizeSecurityGroupIngress(ingressRequest);
  Console.WriteLine("New RDP rule for: " + ipRange);
}
catch (AmazonEC2Exception ex)
{
  // Check the ErrorCode to see if the rule already exists
  if ("InvalidPermission.Duplicate" == ex.ErrorCode)
  {
    Console.WriteLine("An RDP rule for: {0} already exists.", ipRange);
  }
  else
  {
    // The exception was thrown for another reason, so re-throw the exception
    throw;
  }
}
```

Working with Amazon EC2 Key Pairs

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt data, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair. You must specify a key pair when you launch an EC2 instance and specify the private key of the keypair when you connect to the instance. You can create a key pair or use one you've used when launching other instances. For more information, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide for Windows Instances*. This example shows how to create a key pair, describe key pairs and delete a key pair using these AmazonEC2Client methods:

- CreateKeyPair
- DeleteKeyPair
- DescribeKeyPairs

Create a Key Pair and Save the Private Key

When you create a new key pair, you must save the private key that is returned. You cannot retrieve the private key later.

Create and initialize a CreateKeyPairRequest object. Set the `KeyName` property to the name of the key pair.

Pass the request object to the CreateKeyPair method, which returns a CreateKeyPairResult object. If a key pair with the specified name already exists, an AmazonEC2Exception is thrown.

The response object includes a CreateKeyPairResult object that contains the new key's KeyPair object. The KeyPair object's `KeyMaterial` property contains the unencrypted private key for the key pair. Save the private key as a .pem file in a safe location. You'll need this file when you connect to your instance. This example saves the private key in the specified file name.

```
public static void CreateKeyPair(
  AmazonEC2Client ec2Client,
  string keyPairName,
  string privateKeyFile)
{
  var request = new CreateKeyPairRequest();
  request.KeyName = keyPairName;

  try
  {
    var response = ec2Client.CreateKeyPair(request);
    Console.WriteLine();
    Console.WriteLine("New key: " + keyPairName);

    // Save the private key in a .pem file
    using (FileStream s = new FileStream(privateKeyFile, FileMode.Create))
    using (StreamWriter writer = new StreamWriter(s))
    {
      writer.WriteLine(response.KeyPair.KeyMaterial);
    }
  }
  catch (AmazonEC2Exception ex)
  {
    // Check the ErrorCode to see if the key already exists
    if("InvalidKeyPair.Duplicate" == ex.ErrorCode)
    {
      Console.WriteLine("The key pair \"{0}\" already exists.", keyPairName);
    }
    else
    {
      // The exception was thrown for another reason, so re-throw the exception.
      throw;
    }
  }
}

.. _enumerate-key-pairs:
```

Enumerate Your Key Pairs

You can enumerate your key pairs and check whether a key pair exists.

Get the complete list of your key pairs using the DescribeKeyPairs method with no parameters.

```
public static void EnumerateKeyPairs(AmazonEC2Client ec2Client)
{
  var request = new DescribeKeyPairsRequest();
  var response = ec2Client.DescribeKeyPairs(request);

  foreach (KeyPairInfo item in response.KeyPairs)
  {
    Console.WriteLine("Existing key pair: " + item.KeyName);
  }
}
```

```
.. _delete-key-pairs:
```

Delete Key Pairs

You can delete a key pair by calling the DeleteKeyPair from your AmazonEC2Client instance.

Pass a DeleteKeyPairRequest containing the name of the key pair to the DeleteKeyPair method of the AmazonEC2Client object.

```
public static void DeleteKeyPair(
            AmazonEC2Client ec2Client,
            KeyPair keyPair)
{
    try
    {
        // Delete key pair created for sample
        ec2Client.DeleteKeyPair(new DeleteKeyPairRequest { KeyName = keyPair.KeyName });
    }
    catch (AmazonEC2Exception ex)
    {
        // Check the ErrorCode to see if the key already exists
        if ("InvalidKeyPair.NotFound" == ex.ErrorCode)
        {
            Console.WriteLine("The key pair \"{0}\" was not found.", keyPair.KeyName);
        }
        else
        {
            // The exception was thrown for another reason, so re-throw the exception
            throw;
        }
    }
}
```

Launching an Amazon EC2 Instance

Use the following procedure to launch one or more identically configured Amazon EC2 instances from the same Amazon Machine Image (AMI). After you create your EC2 instances, you can check their status. When your EC2 instances are running, you can connect to them.

Launch an EC2 Instance in EC2-Classic or in a VPC

You can launch an instance in either EC2-Classic or in a VPC. For more information about EC2-Classic and EC2-VPC, see Supported Platforms in the *Amazon EC2 User Guide for Windows Instances*.

1. Create and initialize a RunInstancesRequest object. Be sure the AMI, key pair, and security group you specify exist in the region you specified when you created the client object.

   ```
   string amiID = "ami-e189c8d1";
   string keyPairName = "my-sample-key";
   ```

```
List<string> groups = new List<string>() { mySG.GroupId };
var launchRequest = new RunInstancesRequest()
{
  ImageId = amiID,
  InstanceType = InstanceType.T1Micro,
  MinCount = 1,
  MaxCount = 1,
  KeyName = keyPairName,
  SecurityGroupIds = groups
};
```

`ImageId`

    The ID of the AMI. For a list of public AMIs, see Amazon Machine Images.

`InstanceType`

    An instance type that is compatible with the specified AMI. For more information, see Instance Types in the *Amazon EC2 User Guide for Windows Instances*.

`MinCount`

    The minimum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches no instances.

`MaxCount`

    The maximum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches the largest possible number of instances above `MinCount`. You can launch between 1 and the maximum number of instances you're allowed for the instance type. For more information, see How many instances can I run in Amazon EC2 in the Amazon EC2 General FAQ.

`KeyName`

    The name of the EC2 key pair. If you launch an instance without specifying a key pair, you can't connect to it. For more information, see Working with Amazon EC2 Key Pairs.

`SecurityGroupIds`

    The identifiers of one or more security groups. For more information, see Creating a Security Group in Amazon EC2.

2. (Optional) To launch the instance with an IAM role, specify an IAM instance profile in the RunInstancesRequest object.

   An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.

```
{
  "Version": "2012-10-17",
   "Statement": [{
     "Effect": "Allow",
     "Action": [
       "iam:PassRole",
       "iam:ListInstanceProfiles",
       "ec2:*"
```

```
    ],
    "Resource": "*"
  }]
}
```

For example, the following snippet instantiates and configures an IamInstanceProfileSpecification object for an IAM role named `winapp-instance-role-1`.

```
var instanceProfile = new IamInstanceProfile();
instanceProfile.Id  = "winapp-instance-role-1";
```

To specify this instance profile in the RunInstancesRequest object, add the following line.

```
launchRequest.IamInstanceProfile = instanceProfile;
```

3. Launch the instance by passing the request object to the RunInstances method. Save the ID of the instance because you need it to manage the instance.

   Use the returned RunInstancesResponse object to get the instance IDs for the new instances. The `Reservation.Instances` property contains a list of Instance objects, one for each EC2 instance you successfully launched. You can retrieve the ID for each instance from the `InstanceId` property of the Instance object.

```
var launchResponse = ec2Client.RunInstances(launchRequest);
var instances = launchResponse.Reservation.Instances;
var instanceIds = new List<string>();
foreach (Instance item in instances)
{
  instanceIds.Add(item.InstanceId);
  Console.WriteLine();
  Console.WriteLine("New instance: " + item.InstanceId);
  Console.WriteLine("Instance state: " + item.State.Name);
}
```

1. Create and initialize an elastic network interface in a subnet of the VPC.

```
string subnetID = "subnet-cb663da2";

List<string> groups = new List<string>() { mySG.GroupId };
var eni = new InstanceNetworkInterfaceSpecification()
{
  DeviceIndex = 0,
  SubnetId = subnetID,
  Groups = groups,
  AssociatePublicIpAddress = true
};
List<InstanceNetworkInterfaceSpecification> enis = new List<InstanceNetworkInterfaceSpe
```

`DeviceIndex`
: The index of the device on the instance for the network interface attachment.

`SubnetId`
: The ID of the subnet where the instance will be launched.

`GroupIds`
: One or more security groups. For more information, see Creating a Security Group in Amazon EC2.

`AssociatePublicIpAddress`
: Indicates whether to auto-assign a public IP address to an instance in a VPC.

2. Create and initialize a RunInstancesRequest object. Be sure the AMI, key pair, and security group you specify exist in the region you specified when you created the client object.

```
string amiID = "ami-e189c8d1";
string keyPairName = "my-sample-key";

var launchRequest = new RunInstancesRequest()
{
  ImageId = amiID,
  InstanceType = InstanceType.T1Micro,
  MinCount = 1,
  MaxCount = 1,
  KeyName = keyPairName,
  NetworkInterfaces = enis
};
```

`ImageId`
: The ID of the AMI. For a list of public AMIs provided by Amazon, see Amazon Machine Images.

`InstanceType`
: An instance type that is compatible with the specified AMI. For more information, see Instance Types in the *Amazon EC2 User Guide for Windows Instances*.

`MinCount`
: The minimum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches no instances.

`MaxCount`
: The maximum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches the largest possible number of instances above `MinCount`. You can launch between 1 and the maximum number of instances you're allowed for the instance type. For more information, see How many instances can I run in Amazon EC2 in the Amazon EC2 General FAQ.

`KeyName`
: The name of the EC2 key pair. If you launch an instance without specifying a key pair, you can't connect to it. For more information, see Working with Amazon EC2 Key Pairs.

`NetworkInterfaces`
: One or more network interfaces.

3. (Optional) To launch the instance with an IAM role, specify an IAM instance profile in the RunInstancesRequest object.

   An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [{
       "Effect": "Allow",
       "Action": [
         "iam:PassRole",
         "iam:ListInstanceProfiles",
         "ec2:*"
       ],
       "Resource": "*"
     }]
   }
   ```

   For example, the following snippet instantiates and configures an IamInstanceProfileSpecification object for an IAM role named `winapp-instance-role-1`.

   ```
   var instanceProfile = new IamInstanceProfileSpecification();
   instanceProfile.Name  = "winapp-instance-role-1";
   ```

   To specify this instance profile in the RunInstancesRequest object, add the following line.

   ```
   launchRequest.IamInstanceProfile = instanceProfile;
   ```

4. Launch the instances by passing the request object to the RunInstances method. Save the IDs of the instances because you need them to manage the instances.

   Use the returned RunInstancesResponse object to get a list of instance IDs for the new instances. The `Reservation.Instances` property contains a list of Instance objects, one for each EC2 instance you successfully launched. You can retrieve the ID for each instance from the `InstanceId` property of the Instance object'.

   ```
   RunInstancesResponse launchResponse = ec2Client.RunInstances(launchRequest);

   List<String> instanceIds = new List<string>();
   foreach (Instance instance in launchResponse.Reservation.Instances)
   {
     Console.WriteLine(instance.InstanceId);
     instanceIds.Add(instance.InstanceId);
   }
   ```

Check the State of Your Instance

Use the following procedure to get the current state of your instance. Initially, your instance is in the `pending` state. You can connect to your instance after it enters the `running` state.

1. Create and configure a DescribeInstancesRequest object and assign your instance's instance ID to the `InstanceIds` property. You can also use the `Filter` property to limit the request to certain instances, such as instances with a particular user-specified tag.

   ```
   var instanceRequest = new DescribeInstancesRequest();
   instanceRequest.InstanceIds = new List<string>();
   instanceRequest.InstanceIds.Add(instanceId);
   ```

2. Call the DescribeInstances method, and pass it the request object from step 1. The method returns a DescribeInstancesResponse object that contains information about the instance.

   ```
   var response = ec2Client.DescribeInstances(instanceRequest);
   ```

3. The `DescribeInstancesResponse.Reservations` property contains a list of reservations. In this case, there is only one reservation. Each reservation contains a list of `Instance` objects. Again, in this case, there is only one instance. You can get the instance's status from the `State` property.

   ```
   Console.WriteLine(response.Reservations[0].Instances[0].State.Name);
   ```

Connect to Your Running Instance

After an instance is running, you can remotely connect to it by using the appropriate remote client.

For Linux instances, use an SSH client. You must ensure that the instance's SSH port (22) is open to traffic. You will need the instance's public IP address or public DNS name and the private portion of the key pair used to launch the instance. For more information, see Connecting to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.

For Windows instances, use an RDP client. You must ensure the instance's RDP port (3389) is open to traffic. You will need the instance's public IP address or public DNS name and the administrator password. The administrator password is obtained with the GetPasswordData and GetPasswordDataResult.GetDecryptedPassword methods, which require the private portion of the key pair used to launch the instance. For more information, see :ec2-ug-win:`Connecting to Your Windows Instance Using RDP <connecting_to_windows_instance>`in the *Amazon EC2 User Guide for Windows Instances*. The following example demonstrates how to get the password for a Windows instance.

```
public static string GetWindowsPassword(
  AmazonEC2Client ec2Client,
  string instanceId,
  FileInfo privateKeyFile)
{
  string password = "";

  var request = new GetPasswordDataRequest();
  request.InstanceId = instanceId;

  var response = ec2Client.GetPasswordData(request);
  if (null != response.PasswordData)
```

```
  {
    using (StreamReader sr = new StreamReader(privateKeyFile.FullName))
    {
      string privateKeyData = sr.ReadToEnd();
      password = response.GetDecryptedPassword(privateKeyData);
    }
  }
  else
  {
    Console.WriteLine("The password is not available. The password for " +
      "instance {0} is either not ready, or it is not a Windows instance.",
      instanceId);
  }

  return password;
}
```

When you no longer need your EC2 instance, see Terminating an Amazon EC2 Instance.

Terminating an Amazon EC2 Instance

When you no longer need one or more of your Amazon EC2 instances, you can terminate them.

1. Create and initialize a TerminateInstancesRequest object.
2. Set the `TerminateInstancesRequest.InstanceIds` property to a list of one or more instance IDs for the instances to terminate.
3. Pass the request object to the TerminateInstances method. If the specified instance doesn't exist, an AmazonEC2Exception is thrown.
4. You can use the TerminateInstancesResponse object to list the terminated instances, as follows.

```
public static void TerminateInstance(
  AmazonEC2Client ec2Client,
  string instanceId)
{
  var request = new TerminateInstancesRequest();
  request.InstanceIds = new List<string>() { instanceId };

  try
  {
    var response = ec2Client.TerminateInstances(request);
    foreach (InstanceStateChange item in response.TerminatingInstances)
    {
      Console.WriteLine("Terminated instance: " + item.InstanceId);
      Console.WriteLine("Instance state: " + item.CurrentState.Name);
    }
  }
  catch(AmazonEC2Exception ex)
  {
    // Check the ErrorCode to see if the instance does not exist.
```

```
      if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
      {
        Console.WriteLine("Instance {0} does not exist.", instanceId);
      }
      else
      {
        // The exception was thrown for another reason, so re-throw the exception.
        throw;
      }
    }
}
```

Using Regions and Availability Zones with Amazon EC2

This .NET example shows you how to:

- Get details about Availability Zones
- Get details about regions

The Scenario

Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of regions and Availability Zones. Each region is a separate geographic area that has multiple, isolated locations known as Availability Zones. Amazon EC2 provides the ability to place instances and data in multiple locations.

You can use the AWS SDK for .NET to retrieve details about regions and Availability Zones by using the following methods of the AmazonEC2Client class:

- DescribeAvailabilityZones
- DescribeRegions

For more information about regions and Availability Zones, see Regions and Availability Zones in the *Amazon EC2 User Guide for Windows Instances*.

Describe Availability Zones

Create an AmazonEC2Client instance and call the DescribeAvailabilityZones method. The DescribeAvailabilityZonesResponse object that is returned contains a list of Availability Zones.

```
public static void DescribeAvailabilityZones()
{
    Console.WriteLine("Describe Availability Zones");
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeAvailabilityZonesResponse response = client.DescribeAvailabilityZones();
    var availZones = new List<AvailabilityZone>();
    availZones = response.AvailabilityZones;
    foreach (AvailabilityZone az in availZones)
    {
        Console.WriteLine(az.ZoneName);
    }
}
```

Describe Regions

Create an AmazonEC2Client instance and call the DescribeRegions method. The DescribeRegionsResponse object that is returned contains a list of regions.

```
public static void DescribeRegions()
{
    Console.WriteLine("Describe Regions");
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeRegionsResponse response = client.DescribeRegions();
    var regions = new List<Region>();
    regions = response.Regions;
    foreach (Region region in regions)
    {
        Console.WriteLine(region.RegionName);
    }
}
```

Using VPC Endpoints with Amazon EC2

This .NET example shows you how to create, describe, modify, and delete VPC endpoints.

The Scenario

An endpoint enables you to create a private connection between your VPC and another AWS service in your account. You can specify a policy to attach to the endpoint that will control access to the service from your VPC. You can also specify the VPC route tables that use the endpoint.

This example uses the following AmazonEC2Client methods:

- CreateVpcEndpoint
- DescribeVpcEndpoints
- ModifyVpcEndpoint
- DeleteVpcEndpoints

Create a VPC Endpoint

The following example creates a VPC endpoint for an Amazon Simple Storage Service (S3).

Create an AmazonEC2Client instance. You'll create a new VPC so that you can create a VPC endpoint.

Create a CreateVpcRequest object specifying an IPv4 CIDR block as its constructor's parameter. Using that CreateVpcRequest object, use the CreateVpc method to create a VPC. Use that VPC to instantiate a CreateVpcEndpointRequest object, specifying the service name for the endpoint. Then, use that request object to call the CreateVpcEndpoint method and create the VpcEndpoint.

```
public static void CreateVPCEndpoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    CreateVpcRequest vpcRequest = new CreateVpcRequest("10.32.0.0/16");
    CreateVpcResponse vpcResponse = client.CreateVpc(vpcRequest);
    Vpc vpc = vpcResponse.Vpc;
    CreateVpcEndpointRequest endpointRequest = new CreateVpcEndpointRequest();
    endpointRequest.VpcId = vpc.VpcId;
    endpointRequest.ServiceName = "com.amazonaws.us-west-2.s3";
    CreateVpcEndpointResponse cVpcErsp = client.CreateVpcEndpoint(endpointRequest);
    VpcEndpoint vpcEndPoint = cVpcErsp.VpcEndpoint;
}
```

Describe a VPC Endpoint

Create an AmazonEC2Client instance. Next, create a DescribeVpcEndpointsRequest object and limit the maximum number of results to return to 5. Use that `DescribeVpcEndpointsRequest` object to call the DescribeVpcEndpoints method. The DescribeVpcEndpointsResponse that is returned contains the list of VPC Endpoints.

```
public static void DescribeVPCEndPoints()
{
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeVpcEndpointsRequest endpointRequest = new DescribeVpcEndpointsRequest();
    endpointRequest.MaxResults = 5;
    DescribeVpcEndpointsResponse endpointResponse = client.DescribeVpcEndpoints(endpointRec
    List<VpcEndpoint> endpointList = endpointResponse.VpcEndpoints;
    foreach (VpcEndpoint vpc in endpointList)
    {
        Console.WriteLine("VpcEndpoint ID = " + vpc.VpcEndpointId);
        List<string> routeTableIds = vpc.RouteTableIds;
        foreach (string id in routeTableIds)
        {
            Console.WriteLine("\tRoute Table ID = " + id);
        }

    }
}
```

Modify a VPC Endpoint

The following example modifies attributes of a specified VPC endpoint. You can modify the policy associated with the endpoint, and you can add and remove route tables associated with the endpoint.

Create an AmazonEC2Client instance. Create a ModifyVpcEndpointRequest object using the ID of the VPC endpoint and the ID of the route table to add to it. Call the ModifyVpcEndpoint method using the `ModifyVpcEndpointRequest` object. The ModifyVpcEndpointResponse object that is returned contains an HTTP status code indicating whether the modify request succeeded.

```
public static void ModifyVPCEndPoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    ModifyVpcEndpointRequest modifyRequest = new ModifyVpcEndpointRequest();
    modifyRequest.VpcEndpointId = "vpce-17b05a7e";
    modifyRequest.AddRouteTableIds = new List<string> { "rtb-c46f15a3" };
    ModifyVpcEndpointResponse modifyResponse = client.ModifyVpcEndpoint(modifyRequest);
    HttpStatusCode status = modifyResponse.HttpStatusCode;
    if (status.ToString() == "OK")
        Console.WriteLine("ModifyHostsRequest succeeded");
    else
        Console.WriteLine("ModifyHostsRequest failed");
```

Delete a VPC Endpoint

You can delete one or more specified VPC endpoints. Deleting the endpoint also deletes the endpoint routes in the route tables that were associated with the endpoint.

Create an AmazonEC2Client instance. Use the DescribeVpcEndpoints method to list the VPC endpoints associated with the EC2 client. Use the list of VPC endpoints to create a list of VPC endpoint IDs. Use that list to create a DeleteVpcEndpointsRequest object to be used by the DeleteVpcEndpoints method.

```
private static void DeleteVPCEndPoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeVpcEndpointsRequest endpointRequest = new DescribeVpcEndpointsRequest();
    endpointRequest.MaxResults = 5;
    DescribeVpcEndpointsResponse endpointResponse = client.DescribeVpcEndpoints(endpointRec
    List<VpcEndpoint> endpointList = endpointResponse.VpcEndpoints;
    var vpcEndPointListIds = new List<string>();
    foreach (VpcEndpoint vpc in endpointList)
    {
        Console.WriteLine("VpcEndpoint ID = " + vpc.VpcEndpointId);
        vpcEndPointListIds.Add(vpc.VpcEndpointId);
    }
    DeleteVpcEndpointsRequest deleteRequest = new DeleteVpcEndpointsRequest();
    deleteRequest.VpcEndpointIds = vpcEndPointListIds;
    client.DeleteVpcEndpoints(deleteRequest);
}
```

Using Elastic IP Addresses in Amazon EC2

This .NET example shows you how to:

- Retrieve descriptions of your Elastic IP addresses
- Allocate and associate an Elastic IP address with an Amazon EC2 instance
- Release an Elastic IP address

The Scenario

An Elastic IP address is a static IP address designed for dynamic cloud computing. An Elastic IP address is associated with your AWS account, and is a public IP address reachable from the Internet.

If your Amazon EC2 instance doesn't have a public IP address, you can associate an Elastic IP address with your instance to enable communication with the Internet.

In this example, you use the AWS SDK for .NET to manage Elastic IP addresses by using these methods of the Amazon EC2 client class:

- DescribeAddresses
- AllocateAddress
- AssociateAddress
- ReleaseAddress

For more information about Elastic IP addresses in Amazon EC2, see Elastic IP Addresses in the *Amazon EC2 User Guide for Windows Instances*.

Describe Elastic IP Addresses

Create an AmazonEC2Client object. Next, create a DescribeAddressesRequest object to pass as a parameter, filtering the addresses returned by those in your VPC. To retrieve descriptions of all your Elastic IP addresses, omit the filter from the parameters. Then call the DescribeAddresses method of the `AmazonEC2Client` object.

```
public void DescribeElasticIps()
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        var addresses = client.DescribeAddresses(new DescribeAddressesRequest
        {
            Filters = new List<Filter>
            {
                new Filter
                {
                    Name = "domain",
                    Values = new List<string> { "vpc" }
                }
            }
        }).Addresses;

        foreach(var address in addresses)
        {
            Console.WriteLine(address.PublicIp);
            Console.WriteLine("\tAllocation Id: " + address.AllocationId);
            Console.WriteLine("\tPrivate IP Address: " + address.PrivateIpAddress);
            Console.WriteLine("\tAssociation Id: " + address.AssociationId);
            Console.WriteLine("\tInstance Id: " + address.InstanceId);
            Console.WriteLine("\tNetwork Interface Owner Id: " + address.NetworkInterfaceOw
        }
    }
}
```

Allocate and Associate an Elastic IP Address

Create an AmazonEC2Client object. Next, create an AllocateAddressRequest object for the parameter used to allocate an Elastic IP address, which in this case specifies that the domain is a VPC. Call the AllocateAddress method of the `AmazonEC2Client` object.

Upon success, the returned AllocateAddressResponse object has an `AllocationId` property that identifies the allocated Elastic IP address.

Create an AssociateAddressRequest object for the parameters used to associate an Elastic IP address to an Amazon EC2 instance. Include the `AllocationId` from the newly allocated address and the `InstanceId` of the Amazon EC2 instance. Then call the AssociateAddress method of the `AmazonEC2Client` object.

```
public void AllocateAndAssociate(string instanceId)
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        var allocationId = client.AllocateAddress(new AllocateAddressRequest
        {
            Domain = DomainType.Vpc
        }).AllocationId;

        Console.WriteLine("Allocation Id: " + allocationId);

        var associationId = client.AssociateAddress(new AssociateAddressRequest
        {
            AllocationId = allocationId,
            InstanceId = instanceId
        }).AssociationId;

        Console.WriteLine("Association Id: " + associationId);
    }
}
```

Release an Elastic IP Address

Create an AmazonEC2Client object. Next, create a ReleaseAddressRequest object for the parameters used to release an Elastic IP address, which in this case specifies the `AllocationId` for the Elastic IP address. Releasing an Elastic IP address also disassociates it from any Amazon EC2 instance. Call the ReleaseAddress method of the Amazon EC2 service object.

```
public void Release(string allocationId)
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        client.ReleaseAddress(new ReleaseAddressRequest
        {
            AllocationId = allocationId
        });
    }
}
```

Amazon EC2 Spot Instances Examples

This topic describes how to use the AWS SDK for .NET with Amazon EC2 Spot Instances.

Overview

Spot Instances enable you to bid on unused Amazon EC2 capacity and run any instances that you acquire for as long as your bid exceeds the current *Spot Price*. Amazon EC2 changes the Spot Price periodically based on supply and demand; customers whose bids meet or exceed it gain access to the available Spot Instances. Like On-Demand Instances and Reserved Instances, Spot Instances provide another option for obtaining more compute capacity.

Spot Instances can significantly lower your Amazon EC2 costs for applications such as batch processing, scientific research, image processing, video encoding, data and web crawling, financial analysis, and testing. Additionally, Spot Instances are an excellent option when you need large amounts of computing capacity, but the need for that capacity is not urgent.

To use Spot Instances, place a Spot Instance request specifying the maximum price you are willing to pay per instance hour; this is your bid. If your bid exceeds the current Spot Price, your request is fulfilled and your instances will run until either you choose to terminate them or the Spot Price increases above your bid (whichever is sooner). You can terminate a Spot Instance programmatically, as shown this tutorial, or by using the AWS Management Console or by using the AWS Toolkit for Visual Studio.

It's important to note two points:

1. You will often pay less per hour than your bid. Amazon EC2 adjusts the Spot Price periodically as requests come in and available supply changes. Everyone pays the same Spot Price for that period regardless of whether their bid was higher. Therefore, you might pay less than your bid, but you will never pay more than your bid.
2. If you're running Spot Instances and your bid no longer meets or exceeds the current Spot Price, your instances will be terminated. This means you will want to make sure that your workloads and applications are flexible enough to take advantage of this opportunistic—but potentially transient—capacity.

Spot Instances perform exactly like other Amazon EC2 instances while running, and like other Amazon EC2 instances, Spot Instances can be terminated when you no longer need them. If you terminate your instance, you pay for any partial hour used (as you would for On-Demand or Reserved Instances). However, if your instance is terminated by Amazon EC2 because the Spot Price goes above your bid, you will not be charged for any partial hour of use.

This tutorial provides an overview of how to use the .NET programming environment to do the following.

- Submit a Spot request
- Determine when the Spot request becomes fulfilled
- Cancel the Spot request
- Terminate associated instances

Prerequisites

This tutorial assumes you have signed up for AWS, set up your .NET development environment, and installed the AWS SDK for .NET. If you use the Microsoft Visual Studio development environment, we recommend you also install the AWS Toolkit for Visual Studio. For instructions on setting up your environment, see Getting Started with the AWS SDK for .NET.

Setting Up Your Credentials

For information about how to use your AWS credentials with the SDK, see Configuring AWS Credentials.

Submitting Your Spot Request

To submit a Spot request, you first need to determine the instance type, the Amazon Machine Image (AMI), and the maximum bid price you want to use. You must also include a security group, so that you can log into the instance if you want to. For more information about creating security groups, see Creating a Security Group in Amazon EC2.

There are several instance types to choose from; go to Amazon EC2 Instance Types for a complete list. For this tutorial, we will use `t1.micro`. You'll also want to get the ID of a current Windows AMI. For more information, see Finding an AMI in the *Amazon EC2 User Guide for Windows Instances*.

There are many ways to approach bidding for Spot instances. To get a broad overview of the various approaches, you should view the Bidding for Spot Instances video. However, to get started, we'll describe three common strategies: bid to ensure cost is less than on-demand pricing; bid based on the value of the resulting computation; bid so as to acquire computing capacity as quickly as possible.

Reduce Cost Below On-Demand

You have a batch processing job that will take a number of hours or days to run. However, you are flexible with respect to when it starts and ends. You want to see if you can complete it for less than the cost of On-Demand Instances. You examine the Spot Price history for instance types using either the AWS Management Console or the Amazon EC2 API. For more information, go to Viewing Spot Price History. After you've analyzed the price history for your desired instance type in a given Availability Zone, you have two alternative approaches for your bid:

- You could bid at the upper end of the range of Spot Prices (which are still below the On-Demand price), anticipating that your one-time Spot request would most likely be fulfilled and run for enough consecutive compute time to complete the job.
- Or, you could bid at the lower end of the price range, and plan to combine many instances launched over time through a persistent request. The instances would run long enough, in aggregate, to complete the job at an even lower total cost. (We will explain how to automate this task later in this tutorial.)

Pay No More than the Value of the Result

You have a data processing job to run. You understand the value of the job's results well enough to know how much they are worth in terms of computing costs. After you've analyzed the Spot Price history for your instance type, you choose a bid price at which the cost of the computing time is no more than the value of the job's results. You create a persistent bid and allow it to run intermittently as the Spot Price fluctuates at or below your bid.

Acquire Computing Capacity Quickly

You have an unanticipated, short-term need for additional capacity that is not available through On-Demand Instances. After you've analyzed the Spot Price history for your instance type, you bid above the highest historical price to greatly improve the likelihood your request will be fulfilled quickly and continue computing until it is complete.

After you choose your bid price, you are ready to request a Spot Instance. For the purposes of this tutorial, we will set our bid price equal to the On-Demand price ($0.03) to maximize the chances the bid will be fulfilled. You can determine the types of available instances and the On-Demand prices for instances by going to Amazon EC2 Pricing page.

To request a Spot Instance, you need to build your request with the parameters we have specified so far. Start by creating a RequestSpotInstanceRequest object. The request object requires the bid price and the number of instances you want to start. Additionally, you need to set the LaunchSpecification for the request, which includes the instance type, AMI ID, and the name of the security group you want to use for the Spot Instances. After the request is populated, call the RequestSpotInstances method to create the Spot Instance request. The following example demonstrates how to request a Spot Instance.

```
public static SpotInstanceRequest RequestSpotInstance(
  AmazonEC2Client ec2Client,
  string amiId,
  string securityGroupName,
  InstanceType instanceType,
  string spotPrice,
  int instanceCount)
{
  var request = new RequestSpotInstancesRequest();

  request.SpotPrice = spotPrice;
  request.InstanceCount = instanceCount;

  var launchSpecification = new LaunchSpecification();
  launchSpecification.ImageId = amiId;
  launchSpecification.InstanceType = instanceType;

  launchSpecification.SecurityGroups.Add(securityGroupName);

  request.LaunchSpecification = launchSpecification;

  var result = ec2Client.RequestSpotInstances(request);

  return result.SpotInstanceRequests[0];
}
```

The Spot request ID is contained in the `SpotInstanceRequestId` member of the SpotInstanceRequest object.

Running this code will launch a new Spot Instance request.

Note

You will be charged for any Spot Instances that are launched, so make sure you cancel any requests and terminate any instances you launch to reduce any associated fees.

There are other options you can use to configure your Spot requests. To learn more, see RequestSpotInstances in the AWS SDK for .NET.

Determining the State of Your Spot Request

Next, we need to wait until the Spot request reaches the `Active` state before proceeding to the last step. To determine the state of your Spot request, we use the DescribeSpotInstanceRequests method to obtain the state of the Spot request ID we want to monitor.

```
public static SpotInstanceState GetSpotRequestState(
  AmazonEC2Client ec2Client,
  string spotRequestId)
{
  // Create the describeRequest object with all of the request ids
  // to monitor (e.g. that we started).
  var request = new DescribeSpotInstanceRequestsRequest();
  request.SpotInstanceRequestIds.Add(spotRequestId);

  // Retrieve the request we want to monitor.
  var describeResponse = ec2Client.DescribeSpotInstanceRequests(request);

  SpotInstanceRequest req = describeResponse.SpotInstanceRequests[0];

  return req.State;
}
```

Cleaning Up Your Spot Requests and Instances

The final step is to clean up your requests and instances. It is important to both cancel any outstanding requests and terminate any instances. Just canceling your requests will not terminate your instances, which means that you will continue to be charged for them. If you terminate your instances, your Spot requests may be canceled, but there are some scenarios, such as if you use persistent bids, where terminating your instances is not sufficient to stop your request from being re-fulfilled. Therefore, it is a best practice to both cancel any active bids and terminate any running instances.

You use the CancelSpotInstanceRequests method to cancel a Spot request. The following example demonstrates how to cancel a Spot request.

```
public static void CancelSpotRequest(
  AmazonEC2Client ec2Client,
  string spotRequestId)
{
  var cancelRequest = new CancelSpotInstanceRequestsRequest();

  cancelRequest.SpotInstanceRequestIds.Add(spotRequestId);

  ec2Client.CancelSpotInstanceRequests(cancelRequest);
}
```

You use the TerminateInstances method to terminate an instance. The following example demonstrates how to obtain the instance identifier for an active Spot Instance and terminate the instance.

```
public static void TerminateSpotInstance(
  AmazonEC2Client ec2Client,
  string spotRequestId)
{
  var describeRequest = new DescribeSpotInstanceRequestsRequest();
  describeRequest.SpotInstanceRequestIds.Add(spotRequestId);

  // Retrieve the request we want to monitor.
  var describeResponse = ec2Client.DescribeSpotInstanceRequests(describeRequest);

  if (SpotInstanceState.Active == describeResponse.SpotInstanceRequests[0].State)
  {
    string instanceId = describeResponse.SpotInstanceRequests[0].InstanceId;

    var terminateRequest = new TerminateInstancesRequest();
    terminateRequest.InstanceIds = new List<string>() { instanceId };

    try
    {
      var terminateResponse = ec2Client.TerminateInstances(terminateRequest);
    }
    catch (AmazonEC2Exception ex)
    {
      // Check the ErrorCode to see if the instance does not exist.
      if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
      {
        Console.WriteLine("Instance {0} does not exist.", instanceId);
      }
      else
      {
        // The exception was thrown for another reason, so re-throw the exception.
        throw;
      }
    }
  }
}
```

For more information about terminating active instances, see Terminating an Amazon EC2 Instance.

Amazon Glacier Examples

The AWS SDK for .NET supports Amazon Glacier, which is a storage service optimized for infrequently used data, or *cold data*. The service provides durable and extremely low-cost storage with security features for data archiving and backup. For more information, see *Amazon Glacier Developer Guide*.

The following information introduces you to the Amazon Glacier programming models in the AWS SDK for .NET.

Programming Models

The AWS SDK for .NET provides two programming models for working with Amazon Glacier. The following information describes these models and why and how to use them.

Low-Level APIs

The AWS SDK for .NET provides low-level APIs for programming with Amazon Glacier. These low-level APIs map closely to the underlying REST API supported by Amazon Glacier. For each Amazon Glacier REST operation, the low-level APIs provide a corresponding method, a request object for you to provide request information, and a response object for you to process the Amazon Glacier response. The low-level APIs are the most complete implementation of the underlying Amazon Glacier operations.

The following example shows how to use the low-level APIs to list accessible vaults in Amazon Glacier:

```
// using Amazon.Glacier;
// using Amazon.Glacier.Model;

var client = new AmazonGlacierClient();
var request = new ListVaultsRequest();
var response = client.ListVaults(request);

foreach (var vault in response.VaultList)
{
  Console.WriteLine("Vault: {0}", vault.VaultName);
  Console.WriteLine("  Creation date: {0}", vault.CreationDate);
  Console.WriteLine("  Size in bytes: {0}", vault.SizeInBytes);
  Console.WriteLine("  Number of archives: {0}", vault.NumberOfArchives);

  try
  {
    var requestNotifications = new GetVaultNotificationsRequest
    {
      VaultName = vault.VaultName
    };
    var responseNotifications =
      client.GetVaultNotifications(requestNotifications);

    Console.WriteLine("  Notifications:");
    Console.WriteLine("    Topic: {0}",
      responseNotifications.VaultNotificationConfig.SNSTopic);

    var events = responseNotifications.VaultNotificationConfig.Events;

    if (events.Any())
    {
      Console.WriteLine("    Events:");

      foreach (var e in events)
      {
```

```
      Console.WriteLine("{0}", e);
    }
  }
  else
  {
    Console.WriteLine("    No events set.");
  }

}
catch (ResourceNotFoundException)
{
  Console.WriteLine("  No notifications set.");
}

var requestJobs = new ListJobsRequest{
  VaultName = vault.VaultName
};
var responseJobs = client.ListJobs(requestJobs);
var jobs = responseJobs.JobList;

if (jobs.Any())
{
  Console.WriteLine("  Jobs:");

  foreach (var job in jobs)
  {
    Console.WriteLine("    For job ID: {0}",
      job.JobId);
    Console.WriteLine("Archive ID: {0}",
      job.ArchiveId);
    Console.WriteLine("Archive size in bytes: {0}",
      job.ArchiveSizeInBytes.ToString());
    Console.WriteLine("Completed: {0}",
      job.Completed);
    Console.WriteLine("Completion date: {0}",
      job.CompletionDate);
    Console.WriteLine("Creation date: {0}",
      job.CreationDate);
    Console.WriteLine("Inventory size in bytes: {0}",
      job.InventorySizeInBytes);
    Console.WriteLine("Job description: {0}",
      job.JobDescription);
    Console.WriteLine("Status code: {0}",
      job.StatusCode.Value);
    Console.WriteLine("Status message: {0}",
      job.StatusMessage);
  }

}
else
```

```
    {
      Console.WriteLine("  No jobs.");
    }

}
```

For more examples, see:

- Using the AWS SDK for .NET
- Creating a Vault
- Retrieving Vault Metadata
- Downloading a Vault Inventory
- Configuring Vault Notifications
- Deleting a Vault
- Uploading an Archive in a Single Operation
- Uploading Large Archives in Parts
- Downloading an Archive
- Deleting an Archive

For related API reference information, see Amazon.Glacier and Amazon.Glacier.

High-Level APIs

The AWS SDK for .NET provides high-level APIs for programming with Amazon Glacier. To further simplify application development, these high-level APIs offer a higher-level abstraction for some of the operations, including uploading an archive and downloading an archive or vault inventory.

For examples, see the following topics in the *Amazon Glacier Developer Guide*:

- Using the AWS SDK for .NET
- Creating a Vault
- Deleting a Vault
- Upload an Archive to a Vault
- Uploading an Archive
- Uploading Large Archives in Parts
- Download an Archive from a Vault
- Downloading an Archive
- Delete an Archive from a Vault
- Deleting an Archive

For related API reference information, see Amazon.Glacier.Transfer in the *AWS SDK for .NET API Reference*.

AWS Identity and Access Management (IAM) Examples

The AWS SDK for .NET supports IAM, which is a web service that enables AWS customers to manage users and user permissions in AWS.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio a set of C# project templates are installed. So the simplest way to start this project is to open Visual Studio, and then choose **File**, **New Project**, **AWS Sample Projects**, **Deployment and Management**, **AWS Identity and Access Management User**.

For related API reference information, see Amazon.IdentityManagement and Amazon.IdentityManagement.Model.

Prerequisites

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET.

Examples

Managing IAM Account Aliases

These .NET examples show you how to:

- Create an account alias for your AWS account ID
- List an account alias for your AWS account ID
- Delete an account alias for your AWS account ID

The Scenario

If you want the URL for your sign-in page to contain your company name or other friendly identifier instead of your AWS account ID, you can create an alias for your AWS account ID. If you create an AWS account alias, your sign-in page URL changes to incorporate the alias.

The following examples demonstrate how to manage your AWS account alias by using these methods of the AmazonIdentityManagementServiceClient class:

- CreateAccountAlias
- ListAccountAliases
- DeleteAccountAlias

For more information about IAM account aliases, see Your AWS Account ID and Its Alias in the *IAM User Guide*.

Create an Account Alias

Create an AmazonIdentityManagementServiceClient object. Next, create a CreateAccountAliasRequest object containing the new account alias you want to use. Call the CreateAccountAlias method of the `AmazonIAMClient` object. If the account alias is created, display the new alias on the console. If the name already exists, write the exception message to the console.

```
public static void CreateAccountAlias()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new CreateAccountAliasRequest();
        request.AccountAlias = "my-aws-account-alias-2017";
        var response = iamClient.CreateAccountAlias(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.AccountAlias + " created.");
        else
            Console.WriteLine("HTTpStatusCode returned = " + response.HttpStatusCode.ToStr
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

List Account Aliases

Create an AmazonIdentityManagementServiceClient object. Next, create a ListAccountAliasesRequest object. Call the ListAccountAliases method of the `AmazonIAMClient` object. If there is an account alias, display it on the console.

If there is no account alias, write the exception message to the console.

> **Note**
>
> There can be only one account alias.

```
public static void ListAccountAliases()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new ListAccountAliasesRequest();
        var response = iamClient.ListAccountAliases(request);
        List<string> aliases = response.AccountAliases;
        foreach (string account in aliases)
        {
            Console.WriteLine("The account alias is: " + account);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Delete an Account Alias

Create an AmazonIdentityManagementServiceClient object. Next, create a DeleteAccountAliasRequest object containing the account alias you want to delete. Call the DeleteAccountAlias method of the `AmazonIAMClient` object. If the account alias is deleted, display the delete information on the console. If the name doesn't exist, write the exception message to the console.

```
public static void DeleteAccountAlias()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new DeleteAccountAliasRequest();
        request.AccountAlias = "my-aws-account-alias-2017";
        var response = iamClient.DeleteAccountAlias(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.AccountAlias + " deleted.");
        else
            Console.WriteLine("HTTpStatusCode returned = " + response.HttpStatusCode.ToStr
    }
    catch (NoSuchEntityException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Managing IAM Users

This .NET example shows you how to retrieve a list of IAM users, create and delete IAM users, and update an IAM user name.

You can create and manage users in IAM using these methods of the AmazonIdentityManagementServiceClient class:

- CreateUser
- ListUsers
- UpdateUser
- GetUser
- DeleteUser

For more information about IAM users, see IAM Users in the *IAM User Guide*.

For information about limitations on the number of IAM users you can create, see Limitations on IAM Entities in the *IAM User Guide*.

Create a User for Your AWS Account

Create an AmazonIdentityManagementServiceClient object. Next, create a CreateUserRequest object containing the user name you want to use for the new user. Call the CreateUser method of the `AmazonIAMClient` object. If the user name doesn't currently exist, display the name and the ARN for the user on the console. If the name already exists, write a message to that effect to the console.

```
var client = new AmazonIdentityManagementServiceClient();
var request = new CreateUserRequest
{
    UserName = "DemoUser"
};

try
{
    var response = client.CreateUser(request);

    Console.WriteLine("User Name = '{0}', ARN = '{1}'",
      response.User.UserName, response.User.Arn);
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("User 'DemoUser' already exists.");
}
```

List Users in Your AWS Account

This example lists the IAM users that have the specified path prefix. If no path prefix is specified, the action returns all users in the AWS account. If there are no users, the action returns an empty list.

Create an AmazonIdentityManagementServiceClient object. Next, create a ListUsersRequest object containing the parameters needed to list your users. Limit the number returned by setting the `MaxItems` parameter to 10. Call the ListUsers method of the `AmazonIdentityManagementServiceClient` object. Write each user's name and creation date to the console.

```
public static void ListUsers()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestUsers = new ListUsersRequest() { MaxItems = 10 };
    var responseUsers = iamClient.ListUsers(requestUsers);

    foreach (var user in responseUsers.Users)
    {
        Console.WriteLine("User " + user.UserName  + " Created: " + user.CreateDate.ToShort
    }

}
```

Update a User's Name

This example shows how to update the name or the path of the specified IAM user. Be sure you understand the implications of changing an IAM user's path or name. For more information, see Renaming an IAM User in the *IAM User Guide*.

Create an AmazonIdentityManagementServiceClient object. Next, create an UpdateUserRequest object, specifying both the current and new user names as parameters. Call the UpdateUser method of the `AmazonIdentityManagementServiceClient` object.

```
 public static void UpdateUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new UpdateUserRequest
    {
        UserName = "DemoUser",
        NewUserName = "NewUser"
    };

    try
    {
        var response = client.UpdateUser(request);

    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("User 'NewUser' already exists.");
    }
}
```

Get Information about a User

This example shows how to retrieve information about the specified IAM user, including the user's creation date, path, unique ID, and ARN. If you don't specify a user name, IAM determines the user name implicitly based on the AWS access key ID used to sign the request to this API.

Create an AmazonIdentityManagementServiceClient object. Next, create a GetUserRequest object containing the user name you want to get information about. Call the GetUser method of the `AmazonIdentityManagementServiceClient` object to get the information. If the user doesn't exist, an exception is thrown.

```
public static void GetUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new GetUserRequest()
    {
        UserName = "DemoUser"
    };

    try
    {
        var response = client.GetUser(request);
        Console.WriteLine("Creation date: " + response.User.CreateDate.ToShortDateString()
        Console.WriteLine("Password last used: " + response.User.PasswordLastUsed.ToShortDa
        Console.WriteLine("UserId = " + response.User.UserId);

    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine("User DemoUser' does not exist.");
    }
}
```

Delete a User

This example shows how to delete the specified IAM user. The user must not belong to any groups or have any access keys, signing certificates, or attached policies.

Create an AmazonIdentityManagementServiceClient object. Next, create a DeleteUserRequest object containing the parameters needed, which consists of the user name you want to delete. Call the DeleteUser method of the `AmazonIdentityManagementServiceClient` object to delete it. If the user doesn't exist, an exception is thrown.

```
public static void DeleteUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new DeleteUserRequest()
    {
        UserName = "DemoUser"
    };

    try
    {
        var response = client.DeleteUser(request);

    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine("User DemoUser' does not exist.");
    }
}
```

Managing IAM Access Keys

These .NET examples shows you how to:

- Create an access key for a user
- Get the date that an access key was last used
- Update the status for an access key
- Delete an access key

The Scenario

Users need their own access keys to make programmatic calls to AWS from the AWS SDK for .NET. To meet this need, you can create, modify, view, or rotate access keys (access key IDs and secret access keys) for IAM users. When you create an access key, its status is Active by default, which means the user can use the access key for API calls.

The C# code uses the AWS SDK for .NET to manage IAM access keys using these methods of the AmazonIdentityManagementServiceClient class:

- CreateAccessKey
- ListAccessKeys
- GetAccessKeyLastUsed
- UpdateAccessKey
- DeleteAccessKey

For more information about IAM access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*.

Create Access Keys for a User

Call the `CreateAccessKey` method to create an access key named `S3UserReadOnlyAccess` for the IAM access keys examples. The `CreateAccessKey method first creates a user named :code:`S3UserReadOnlyAccess` with read only access rights by calling the `CreateUser` method. It then creates an AmazonIdentityManagementServiceClient object and a CreateAccessKeyRequest object containing the `UserName` parameter needed to create new access keys. It then calls the CreateAccessKey method of the `AmazonIdentityManagementServiceClient` object.

```
public static void CreateAccessKey()
{
    try
    {
        CreateUser();
        var iamClient = new AmazonIdentityManagementServiceClient();
        // Create an access key for the IAM user that can be used by the SDK
        var accessKey = iamClient.CreateAccessKey(new CreateAccessKeyRequest
        {
            // Use the user created in the CreateUser example
            UserName = "S3UserReadOnlyAccess"
        }).AccessKey;

    }
    catch (LimitExceededException e)
    {
        Console.WriteLine(e.Message);
    }
}

public static User CreateUser()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    try
    {
        // Create the IAM user
        var readOnlyUser = iamClient.CreateUser(new CreateUserRequest
        {
            UserName = "S3UserReadOnlyAccess"
        }).User;
```

```
            // Assign the read-only policy to the new user
            iamClient.PutUserPolicy(new PutUserPolicyRequest
            {
                UserName = readOnlyUser.UserName,
                PolicyName = "S3ReadOnlyAccess",
                PolicyDocument = S3_READONLY_POLICY
            });
            return readOnlyUser;
        }
        catch (EntityAlreadyExistsException e)
        {
            Console.WriteLine(e.Message);
            var request = new GetUserRequest()
            {
                UserName = "S3UserReadOnlyAccess"
            };

            return iamClient.GetUser(request).User;

        }
    }
```

List a User's Access Keys

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the parameters needed to retrieve the user's access keys. This includes the IAM user's name and, optionally, the maximum number of access key pairs you want to list. Call the ListAccessKeys method of the `AmazonIdentityManagementServiceClient` object.

```
public static void ListAccessKeys()
{

    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        // Use the user created in the CreateAccessKey example
        UserName = "S3UserReadOnlyAccess",
        MaxItems = 10
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    Console.WriteLine("  Access keys:");

    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
        Console.WriteLine("    {0}", accessKey.AccessKeyId);
     }
}
```

Get the Last Used Date for Access Keys

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the `UserName` parameter needed to list the access keys. Call the ListAccessKeys method of the `AmazonIdentityManagementServiceClient` object. Loop through the access keys returned, displaying the `AccessKeyId` of each key and using it to create a GetAccessKeyLastUsedRequest object. Call the GetAccessKeyLastUsed method and display the time that the key was last used on the console.

```
public static void GetAccessKeysLastUsed()
{

    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        // Use the user we created in the CreateUser example
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    Console.WriteLine("  Access keys:");

    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
        Console.WriteLine("    {0}", accessKey.AccessKeyId);
        GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest()
            { AccessKeyId = accessKey.AccessKeyId };
        var response = iamClient.GetAccessKeyLastUsed(request);
        Console.WriteLine("Key last used " + response.AccessKeyLastUsed.LastUsedDate.ToLong
    }
}
```

Update the Status of an Access Key

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the user name to list the keys for. The user name in this example is the user created for the other examples. Call the ListAccessKeys method of the `AmazonIdentityManagementServiceClient`. The ListAccessKeysResponse that is returned contains a list of the access keys for that user. Use the first access key in the list. Create an UpdateAccessKeyRequest object, providing the `UserName`, `AccessKeyId`, and `Status` parameters. Call the UpdateAccessKey method of the `AmazonIdentityManagementServiceClient` object.

```
public static void UpdateKeyStatus()
{
    // This example changes the status of the key specified by its index in the list of access keys
    // Optionally, you could change the keynumber parameter to be an AccessKey ID
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    UpdateAccessKeyRequest updateRequest = new UpdateAccessKeyRequest
        {
            UserName = "S3UserReadOnlyAccess",
            AccessKeyId = responseAccessKeys.AccessKeyMetadata[0].AccessKeyId,
            Status = StatusType.Active
        };
    iamClient.UpdateAccessKey(updateRequest);
    Console.WriteLine("  Access key " + updateRequest.AccessKeyId + " updated");
}
```

Delete Access Keys

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the name of the user as a parameter. Call the ListAccessKeys method of the `AmazonIdentityManagementServiceClient`. The ListAccessKeysResponse that is returned contains a list of the access keys for that user. Delete each access key in the list by calling the DeleteAccessKey method of the `AmazonIdentityManagementServiceClient`.

```
public static void DeleteAccessKeys()
{
// Delete all the access keys created for the examples
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        // Use the user created in the CreateUser example
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    Console.WriteLine("  Access keys:");

    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
        Console.WriteLine("    {0}", accessKey.AccessKeyId);
        iamClient.DeleteAccessKey(new DeleteAccessKeyRequest
        {
            UserName = "S3UserReadOnlyAccess",
            AccessKeyId = accessKey.AccessKeyId
        });
        Console.WriteLine("Access Key " + accessKey.AccessKeyId + " deleted");
    }

}
```

Working with IAM Policies

The following examples show you how to:

- Create and delete IAM policies
- Attach and detach IAM policies from roles

The Scenario

You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources those actions can affect. Any actions or resources that are not explicitly allowed are denied by default. You can create policies and attach them to users, groups of users, roles assumed by users, and resources.

Use the AWS SDK for .NET to create and delete policies and attach and detach role policies by using these methods of the AmazonIdentityManagementServiceClient class:

- CreatePolicy
- GetPolicy
- ListAttachedRolePolicies
- AttachRolePolicy
- DetachRolePolicy

For more information about IAM users, see Overview of Access Management: Permissions and Policies in the *IAM User Guide*.

Create an IAM Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a CreatePolicy object containing the parameters needed to create a new policy, which consists of the name you want to use for the new policy and a policy document. You create the policy document by calling the provided `GenerateRolePolicyDocument` method. Upon returning from the `CreatePolicy` method call, the CreatePolicyResponse contains the policy ARN, which is displayed on the console. Please make a note of it so you can use it in the following examples.

```
public static void CreatePolicyExample()
{
    var client = new AmazonIdentityManagementServiceClient();
    // GenerateRolePolicyDocument is a custom method
    string policyDoc = GenerateRolePolicyDocument();

    var request = new CreatePolicyRequest
    {
        PolicyName = "DemoEC2Permissions",
        PolicyDocument = policyDoc
    };

    try
    {
        var createPolicyResponse = client.CreatePolicy(request);
        Console.WriteLine("Make a note, Policy named " + createPolicyResponse.Policy.Policy
            " has Arn: : " + createPolicyResponse.Policy.Arn);
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine
          ("Policy 'DemoEC2Permissions' already exits.");
    }

}
```

```
public static string GenerateRolePolicyDocument()
{
    // using Amazon.Auth.AccessControlPolicy;

    // Create a policy that looks like this:
    /*
  {
    "Version" : "2012-10-17",
    "Id"  : "DemoEC2Permissions",
    "Statement" : [
      {
        "Sid" : "DemoEC2PermissionsStatement",
        "Effect" : "Allow",
        "Action" : [
          "s3:Get*",
          "s3:List*"
        ],
        "Resource" : "*"
      }
    ]
  }
  */

    var actionGet = new ActionIdentifier("s3:Get*");
    var actionList = new ActionIdentifier("s3:List*");
    var actions = new List<ActionIdentifier>();

    actions.Add(actionGet);
    actions.Add(actionList);

    var resource = new Resource("*");
    var resources = new List<Resource>();

    resources.Add(resource);

    var statement = new Amazon.Auth.AccessControlPolicy.Statement(Amazon.Auth.AccessControl
    {
        Actions = actions,
        Id = "DemoEC2PermissionsStatement",
        Resources = resources
    };
    var statements = new List<Amazon.Auth.AccessControlPolicy.Statement>();

    statements.Add(statement);

    var policy = new Policy
    {
        Id = "DemoEC2Permissions",
        Version = "2012-10-17",
        Statements = statements
    };

    return policy.ToJson();
}
```

Get an IAM Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a GetPolicyRequest object containing the parameter needed to get the policy, the policy ARN, which was returned by the `CreatePolicy` method in the previous example.

Call the GetPolicy method.

```
public static void GetPolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new GetPolicyRequest
    {
        PolicyArn = "arn:aws:iam::123456789:policy/DemoEC2Permissions"
    };

    try
    {
        var response = client.GetPolicy(request);
        Console.WriteLine("Policy " + response.Policy.PolicyName + "successfully retrieved"

    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine
          ("Policy 'DemoEC2Permissions' does not exist.");
    }

}
```

Attach a Managed Role Policy

Create an AmazonIdentityManagementServiceClient object. Next, create an AttachRolePolicy object containing the parameters needed to attach the policy to the role, the role name, and the Jason policy returned by the `GenerateRolePolicyDocument` method. Be sure to use a valid role from the roles associated with your AWS account.

```
public static void AttachRolePolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    string policy = GenerateRolePolicyDocument();
    CreateRoleRequest roleRequest = new CreateRoleRequest()
    {
        RoleName = "tester",
        AssumeRolePolicyDocument = policy
    };

    var request = new AttachRolePolicyRequest()
    {
        PolicyArn = "arn:aws:iam::123456789:policy/DemoEC2Permissions",
        RoleName = "tester"
    };
    try
    {
        var response = client.AttachRolePolicy(request);
        Console.WriteLine("Policy DemoEC2Permissions attached to Role TestUser");
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine
          ("Policy 'DemoEC2Permissions' does not exist");
    }
    catch (InvalidInputException)
    {
        Console.WriteLine
          ("One of the parameters is incorrect");
    }
}
```

Detach a Managed Role Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a DetachRolePolicy object containing the parameters needed to attach the policy to the role, the role name, and the Jason policy returned by the `GenerateRolePolicyDocument` method. Be sure to use the role you used to attach the policy in the previous example.

```
public static void DetachRolePolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    string policy = GenerateRolePolicyDocument();
    CreateRoleRequest roleRequest = new CreateRoleRequest()
    {
        RoleName = "tester",
        AssumeRolePolicyDocument = policy
    };

    var request = new DetachRolePolicyRequest()
    {
        PolicyArn = "arn:aws:iam::123456789:policy/DemoEC2Permissions",
        RoleName = "tester"
    };
    try
    {
        var response = client.DetachRolePolicy(request);
        Console.WriteLine("Policy DemoEC2Permissions detached from Role 'tester'");
    }
    catch (NoSuchEntityException e)
    {
        Console.WriteLine
          (e.Message);
    }
    catch (InvalidInputException i)
    {
        Console.WriteLine
          (i.Message);
    }
}
```

Working with IAM Server Certificates

These .NET examples show you how to:

- List server certificates
- Get server certificates
- Update server certificates
- Delete server certificates

The Scenario

In these, examples, you'll basic tasks for managing server certificates for HTTPS connections. To enable HTTPS connections to your website or application on AWS, you need an SSL/TLS server certificate. To use a certificate that you obtained from an external provider with your website or application on AWS, you must upload the certificate to IAM or import it into AWS Certificate Manager.

These examples use the AWS SDK for .NET to send and receive messages by using these methods of the AmazonIdentityManagementServiceClient class:

- ListServerCertificates
- GetServerCertificate
- UpdateServerCertificate
- DeleteServerCertificate

For more information about server certificates, see Working with Server Certificates in the *IAM User Guide*.

List Your Server Certificates

Create an AmazonIdentityManagementServiceClient object. Next, create a ListServerCertificatesRequest object.

There are no required parameters. Call the ListServerCertificates method of the `AmazonIdentityManagementServiceClient` object.

```
public static void ListCertificates()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new ListServerCertificatesRequest();
        var response = iamClient.ListServerCertificates(request);
        foreach (KeyValuePair<string, string> kvp in response.ResponseMetadata.Metadata)
        {
            Console.WriteLine("Key = {0}, Value = {1}",
                kvp.Key, kvp.Value);
        }
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Get a Server Certificate

Create an AmazonIdentityManagementServiceClient object. Next, create a GetServerCertificateRequest object, specifying the `ServerCertificateName`. Call the GetServerCertificate method of the `AmazonIdentityManagementServiceClient` object.

```
public static void GetCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new GetServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        var response = iamClient.GetServerCertificate(request);
        Console.WriteLine("CertificateName = " + response.ServerCertificate.ServerCertifica
        Console.WriteLine("Certificate Arn = " + response.ServerCertificate.ServerCertifica
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Update a Server Certificate

Create an AmazonIdentityManagementServiceClient object. Next, create an UpdateServerCertificateRequest object, specifying the `ServerCertificateName` and the `NewServerCertificateName`. Call the UpdateServerCertificate method of the `AmazonIdentityManagementServiceClient` object.

```
public static void UpdateCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new UpdateServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        request.NewServerCertificateName = "NEW_Certificate_NAME";
        var response = iamClient.UpdateServerCertificate(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine("Update succesful");
        else
            Console.WriteLine("HTTpStatusCode returned = " + response.HttpStatusCode.ToStr
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }

}
```

Delete a Server Certificate

Create an AmazonIdentityManagementServiceClient object. Next, create a DeleteServerCertificateRequest object, specifying the `ServerCertificateName`. Call the DeleteServerCertificate method of the `AmazonIdentityManagementServiceClient` object.

```
public static void DeleteCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new DeleteServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        var response = iamClient.DeleteServerCertificate(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.ServerCertificateName + " deleted");
        else
            Console.WriteLine("HTTpStatusCode returned = " + response.HttpStatusCode.ToStr
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

List IAM Account Information

The AWS SDK for .NET supports IAM, which is a web service that enables AWS customers to manage users and user permissions in AWS.

The following example shows how to use the low-level APIs to list accessible user accounts in IAM. For each user account, its associated groups, policies, and access key IDs are also listed.

```
public static void ListUsers()
{
  var iamClient = new AmazonIdentityManagementServiceClient();
  var requestUsers = new ListUsersRequest();
  var responseUsers = iamClient.ListUsers(requestUsers);

  foreach (var user in responseUsers.Users)
  {
    Console.WriteLine("For user {0}:", user.UserName);
    Console.WriteLine("  In groups:");

    var requestGroups = new ListGroupsForUserRequest
    {
      UserName = user.UserName
    };
    var responseGroups = iamClient.ListGroupsForUser(requestGroups);

    foreach (var group in responseGroups.Groups)
    {
      Console.WriteLine("    {0}", group.GroupName);
    }

    Console.WriteLine("  Policies:");

    var requestPolicies = new ListUserPoliciesRequest
    {
      UserName = user.UserName
    };
    var responsePolicies = iamClient.ListUserPolicies(requestPolicies);

    foreach (var policy in responsePolicies.PolicyNames)
    {
      Console.WriteLine("    {0}", policy);
    }

    var requestAccessKeys = new ListAccessKeysRequest
    {
      UserName = user.UserName
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);

    Console.WriteLine("  Access keys:");

    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
      Console.WriteLine("    {0}", accessKey.AccessKeyId);
    }
    }
}
```

For related API reference information, see Amazon.IdentityManagement and Amazon.IdentityManagement.Model.

Granting Access Using an IAM Role

This .NET example shows you how to:

- Create a sample program that retrieves an object from Amazon S3
- Create an IAM role
- Launch an Amazon EC2 instance and specify the IAM role
- Run the sample on the Amazon EC2 instance

The Scenario

All requests to AWS must be cryptographically signed by using credentials issued by AWS. Therefore, you need a strategy to manage credentials for software that runs on Amazon EC2 instances. You have to distribute, store, and rotate these credentials securely, but also keep them accessible to the software.

IAM roles enable you to effectively manage AWS credentials for software running on EC2 instances. You create an IAM role and configure it with the permissions the software requires. For more information about the benefits of using IAM roles, see IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide for Windows Instances* and Roles (Delegation and Federation) in the *IAM User Guide*.

To use the permissions, the software constructs a client object for the AWS service. The constructor searches the credentials provider chain for credentials. For .NET, the credentials provider chain is as follows:

- The App.config file
- The instance metadata associated with the IAM role for the EC2 instance

If the client doesn't find credentials in App.config, it retrieves temporary credentials that have the same permissions as those associated with the IAM role from instance metadata. The credentials are stored by the constructor on behalf of the application software, and are used to make calls to AWS from that client object. Although the credentials are temporary and eventually expire, the SDK client periodically refreshes them so that they continue to enable access. This periodic refresh is completely transparent to the application software.

The following examples show a sample program that retrieves an object from Amazon S3 using the AWS credentials you configure. You create an IAM role to provide the AWS credentials. Finally, you launch an instance with an IAM role that provides the AWS credentials to the sample program running on the instance.

Create a Sample that Retrieves an Object from Amazon S3

The following sample code requires a text file in an Amazon S3 bucket that you have access to, and AWS credentials that provide you with access to the Amazon S3 bucket.

For more information about creating an Amazon S3 bucket and uploading an object, see the *Amazon S3 Getting Started Guide*. For more information about AWS credentials, see Configuring AWS Credentials.

```
using System;
using System.Collections.Specialized;
using System.IO;

using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

namespace Aws3Sample
{
  class S3Sample
  {
    public static void Main(string[] args)
    {
      ReadS3File("bucket-name", "s3-file-name", "output-file-name");

      Console.WriteLine("Press enter to continue");
      Console.ReadLine();
    }

    public static void ReadS3File(
      string bucketName,
      string keyName,
      string filename)
    {

      string responseBody = "";

      try
      {
        using (var s3Client = new AmazonS3Client())
        {
          Console.WriteLine("Retrieving (GET) an object");

          var request = new GetObjectRequest()
          {
            BucketName = bucketName,
            Key = keyName
          };

          using (var response = s3Client.GetObject(request))
          using (var responseStream = response.ResponseStream)
          using (var reader = new StreamReader(responseStream))
          {
            responseBody = reader.ReadToEnd();
          }
        }

        using (var s = new FileStream(filename, FileMode.Create))
        using (var writer = new StreamWriter(s))
        {
          writer.WriteLine(responseBody);
        }
      }
      catch (AmazonS3Exception s3Exception)
      {
        Console.WriteLine(s3Exception.Message, s3Exception.InnerException);
      }
    }
  }
}
```

1. Open Visual Studio and create an AWS Console project.
2. Add the AWSSDK.S3 package to your project.
3. Replace the code in the `Program.cs` file with the sample code.
4. Replace the following values:

   `bucket-name`

   The name of your Amazon S3 bucket.

   `s3-file-name`

   The path and name of a text file in the bucket.

   `output-file-name`

   The path and file name to write the file to.
5. Compile and run the sample program. If the program succeeds, it displays the following output and creates a file named `s3Object.txt` on your local drive that contains the text it retrieved from the text file in Amazon S3.

   ```
   Retrieving (GET) an object
   ```

   If the program fails, be sure you're using credentials that provide you with access to the bucket.
6. (Optional) Transfer the sample program to a running Windows instance on which you haven't set up credentials. Run the program and verify that it fails because it can't locate credentials.

Create an IAM Role

Create an IAM role that has the appropriate permissions to access Amazon S3.

1. Open the IAM console.
2. In the navigation pane, choose **Roles**, and then choose **Create New Role**.
3. Type a name for the role, and then choose **Next Step**. Remember this name because you'll need it when you launch your EC2 instance.
4. Under **AWS Service Roles**, choose **Amazon EC2**. Under **Select Policy Template**, choose **Amazon S3 Read Only Access**. Review the policy, and then choose **Next Step**.
5. Review the role information, and then choose **Create Role**.

Launch an EC2 Instance and Specify the IAM Role

You can use the Amazon EC2 console or the AWS SDK for .NET to launch an EC2 instance with an IAM role.

- Using the console: Follow the directions in Launching a Windows Instance in the *Amazon EC2 User Guide for Windows Instances*. When you reach the **Review Instance Launch** page, choose **Edit instance details**. In **IAM role**, specify the IAM role you created previously. Complete the procedure as directed. You'll need to create or use an existing security group and key pair to connect to the instance.
- Using the AWS SDK for .NET: See Launching an Amazon EC2 Instance.

An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:PassRole",
      "iam:ListInstanceProfiles",
      "ec2:*"
    ],
    "Resource": "*"
  }]
}
```

Run the Sample Program on the EC2 Instance

To transfer the sample program to your EC2 instance, connect to the instance using the AWS Management Console as described in the following procedure.

Note

Alternatively, connect using the Toolkit for Visual Studio (see Connecting to an Amazon EC2 Instance in the AWS Toolkit for Visual Studio) and then copy the files from your local drive to the instance. The Remote Desktop session is automatically configured so that your local drives are available to the instance.

1. Open the Amazon EC2 console.
2. Get the password for your EC2 instance:
    1. In the navigation pane, choose **Instances**. Choose the instance, and then choose **Connect**.

    1. In the **Connect To Your Instance** dialog box, choose **Get Password**. (It will take a few minutes after the instance is launched before the password is available.)
    2. Choose **Browse** and navigate to the private key file you created when you launched the instance. Choose the file, and then choose **Open** to copy the file's contents into the contents box.

   3. Choose **Decrypt Password**. The console displays the default administrator password for the instance in the **Connect To Your Instance** dialog box, replacing the link to **Get Password** shown earlier with the password.

   4. Record the default administrator password or copy it to the clipboard. You need this password to connect to the instance.

3. Connect to your EC2 instance:

   1. Choose **Download Remote Desktop File**. When your browser prompts you, save the `.rdp` file. When you finish, choose **Close** to close the **Connect To Your Instance** dialog box.
   2. Navigate to your downloads directory, right-click the `.rdp` file, and then choose **Edit**. On the **Local Resources** tab, under **Local devices and resources**, choose **More**. Choose **Drives** to make your local drives available to your instance. Then choose **OK**.

   3. Choose **Connect** to connect to your instance. You may get a warning that the publisher of the remote connection is unknown.

   4. Sign in to the instance when prompted, using the default **Administrator** account and the default administrator password you recorded or copied previously.

      Sometimes copying and pasting content can corrupt data. If you encounter a "Password Failed" error when you sign in, try typing in the password manually. For more information, see Connecting to Your Windows Instance Using RDP and Troubleshooting Windows Instances in the *Amazon EC2 User Guide for Windows Instances*.

4. Copy the program and the AWS assemblies (`AWSSDK.Core.dll` and `AWSSDK.S3.dll`) from your local drive to the instance.

5. Run the program and verify that it succeeds using the credentials provided by the IAM role.

   ```
   Retrieving (GET) an object
   ```

Amazon Route 53 Examples

The AWS SDK for .NET supports Amazon Route 53, which is a Domain Name System (DNS) web service that provides secure and reliable routing to your infrastructure that uses Amazon Web Services (AWS) products, such as Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing, or Amazon Simple Storage Service (Amazon S3). You can also use Amazon Route 53 to route users to your infrastructure outside of AWS. This topic describes how to use the AWS SDK for .NET to create an Amazon Route 53 hosted zone and add a new resource record set to that zone.

Note

This topic assumes you are already familiar with how to use Amazon Route 53 and have already installed the AWS SDK for .NET. For more information about Amazon Route 53, see the Amazon Route 53 Developer Guide. For information about how to install the AWS SDK for .NET, see Getting Started with the AWS SDK for .NET.

The basic procedure is as follows.

To create a hosted zone and update its record sets

1. Create a hosted zone.
2. Create a change batch that contains one or more record sets, and instructions on which action to take for each set.
3. Submit a change request to the hosted zone that contains the change batch.
4. Monitor the change to verify it is complete.

The example is a simple console application that shows how to use the AWS SDK for .NET to implement this procedure for a basic record set.

To run this example

1. In the Visual Studio **File** menu, choose **New**, and then choose **Project**.
2. Choose the **AWS Empty Project** template and specify the project's name and location.
3. Specify the application's default credentials profile and AWS region, which are added to the project's App.config file. This example assumes the region is set to US East (N. Virginia) and the profile is set to default. For more information on profiles, see Configuring AWS Credentials.
4. Open program.cs and replace the `using` declarations and the code in `Main` with the corresponding code from the following example. If you are using your default credentials profile and region, you can compile and run the application as-is. Otherwise, you must provide an appropriate profile and region, as discussed in the notes that follow the example.

```
using System;
using System.Collections.Generic;
using System.Threading;

using Amazon;
using Amazon.Route53;
using Amazon.Route53.Model;

namespace Route53_RecordSet
{
  //Create a hosted zone and add a basic record set to it
  class recordset
  {
    public static void Main(string[] args)
    {
      string domainName = "www.example.org";

      //[1] Create an Amazon Route 53 client object
      var route53Client = new AmazonRoute53Client();

      //[2] Create a hosted zone
      var zoneRequest = new CreateHostedZoneRequest()
      {
        Name = domainName,
        CallerReference = "my_change_request"
      };
      var zoneResponse = route53Client.CreateHostedZone(zoneRequest);

      //[3] Create a resource record set change batch
      var recordSet = new ResourceRecordSet()
      {
        Name = domainName,
        TTL = 60,
        Type = RRType.A,
        ResourceRecords = new List<ResourceRecord>
        {
          new ResourceRecord { Value = "192.0.2.235" }
        }
      };

      var change1 = new Change()
      {
        ResourceRecordSet = recordSet,
        Action = ChangeAction.CREATE
      };

      var changeBatch = new ChangeBatch()
      {
        Changes = new List<Change> { change1 }
      };

      //[4] Update the zone's resource record sets
      var recordsetRequest = new ChangeResourceRecordSetsRequest()
      {
        HostedZoneId = zoneResponse.HostedZone.Id,
        ChangeBatch = changeBatch
      };

      var recordsetResponse = route53Client.ChangeResourceRecordSets(recordsetRequest);

      //[5] Monitor the change status
      var changeRequest = new GetChangeRequest()
      {
        Id = recordsetResponse.ChangeInfo.Id
      };

      while (ChangeStatus.PENDING ==
        route53Client.GetChange(changeRequest).ChangeInfo.Status)
      {
        Console.WriteLine("Change is pending.");
        Thread.Sleep(15000);
      }

      Console.WriteLine("Change is complete.");
      Console.ReadKey();
    }
    }
}
```

The numbers in the following sections are keyed to the comments in the preceding example.

[1] Create a Client Object

The object must have the following information:

An AWS region

When you call a client method, the underlying HTTP request is sent to this endpoint.

A credentials profile

The profile must grant permissions for the actions you intend to use—the Amazon Route 53 actions in this case. Attempts to call actions that lack permissions will fail. For more information, see Configuring AWS Credentials.

The AmazonRoute53Client class supports a set of public methods that you use to invoke Amazon Route 53 actions. You create the client object by instantiating a new instance of the `AmazonRoute53Client` class. There are multiple constructors.

[2] Create a hosted zone

A hosted zone serves the same purpose as a traditional DNS zone file. It represents a collection of resource record sets that are managed together under a single domain name.

To create a hosted zone

1. Create a CreateHostedZoneRequest object and specify the following request parameters. There are also two optional parameters that aren't used by this example.

   `Name`

   (Required) The domain name you want to register, `www.example.com` for this example. This domain name is intended only for examples. It can't be registered with a domain name registrar, but you can use it to create a hosted zone for learning purposes.

   `CallerReference`

   (Required) An arbitrary user-defined string that serves as a request ID and can be used to retry failed requests. If you run this application multiple times, you must change the `CallerReference` value.

2. Pass the `CreateHostedZoneRequest` object to the client object's CreateHostedZone method. The method returns a CreateHostedZoneResponse object that contains information about the request, including the HostedZone.Id property that identifies zone.

[3] Create a resource record set change batch

A hosted zone can have multiple resource record sets. Each set specifies how a subset of the domain's traffic, such as email requests, should be routed. You can update a zone's resource record sets with a single request. The first step is to package all the updates in a ChangeBatch object. This example specifies only one update, adding a basic resource record set to the zone, but a `ChangeBatch` object can contain updates for multiple resource record sets.

To create a ChangeBatch object

1. Create a ResourceRecordSet object for each resource record set you want to update. The group of properties you specify depends on the type of resource record set. For a complete description of the properties used by the different resource record sets, see Values that You Specify When You

Create or Edit Amazon Route 53 Resource Record Sets. The example `ResourceRecordSet` object represents a basic resource record set , and specifies the following required properties.

`Name`

The domain or subdomain name, `www.example.com` for this example.

`TTL`

The amount of time, in seconds, the DNS recursive resolvers should cache information about this resource record set, 60 seconds for this example.

`Type`

The DNS record type, `A` for this example. For a complete list, see Supported DNS Resource Record Types.

`ResourceRecords`

A list of one or more ResourceRecord objects, each of which contains a DNS record value that depends on the DNS record type. For an `A` record type, the record value is an IPv4 address, which for this example is set to a standard example address, `192.0.2.235`.

2. Create a Change object for each resource record set, and set the following properties.

   `ResourceRecordSet`

   The `ResourceRecordSet` object you created in the previous step.

   `Action`

   The action to be taken for this resource record set: `CREATE`, `DELETE`, or `UPSERT`. For more information about these actions, see Elements. This example creates a new resource record set in the hosted zone, so `Action` is set to `CREATE`.

3. Create a ChangeBatch object and set its `Changes` property to a list of the `Change` objects that you created in the previous step.

[4] Update the zone's resource record sets

To update the resource record sets, pass the `ChangeBatch` object to the hosted zone, as follows.

To update a hosted zone's resource record sets

1. Create a ChangeResourceRecordSetsRequest object with the following property settings.

   `HostedZoneId`

   The hosted zone's ID, which the example sets to the ID that was returned in the `CreateHostedZoneResponse` object. To get the ID of an existing hosted zone, call ListHostedZones.

   `ChangeBatch`

   A `ChangeBatch` object that contains the updates.

2. Pass the `ChangeResourceRecordSetsRequest` object to the ChangeResourceRecordSets method of the client object. It returns a ChangeResourceRecordSetsResponse object, which contains a request ID you can use to monitor the request's progress.

[5] Monitor the update status

Resource record set updates typically take a minute or so to propagate through the system. You can monitor the update's progress and verify that it is complete as follows.

To monitor update status

1. Create a GetChangeRequest object and set its `Id` property to the request ID that was returned by `ChangeResourceRecordSets`.
2. Use a wait loop to periodically call the GetChange method of the client object. `GetChange` returns `PENDING` while the update is in progress and `INSYNC` after the update is complete. You can use the same `GetChangeRequest` object for all of the method calls.

Amazon Simple Storage Service Programming with the AWS SDK for .NET

The AWS SDK for .NET supports Amazon Simple Storage Service (Amazon S3), which is storage for the Internet. It is designed to make web-scale computing easier for developers. For more information, see Amazon S3.

The following links provide examples of programming Amazon S3 with the AWS SDK for .NET:

- Using the AWS SDK for .NET for Amazon S3 Programming
- Making Requests Using AWS Account or IAM User Credentials
- Making Requests Using IAM User Temporary Credentials
- Making Requests Using Federated User Temporary Credentials
- Managing ACLs
- Creating a Bucket
- Upload an Object
- Multipart Upload with the High-Level API
- Multipart Upload with the Low-Level API
- Listing Objects
- Listing Keys
- Get an Object
- Copy an Object
- Copy an Object with the Multipart Upload API
- Deleting an Object
- Deleting Multiple Objects
- Restore an Object
- Configure a Bucket for Notifications
- Manage an Object's Lifecycle
- Generate a Pre-signed Object URL
- Managing Websites
- Enabling Cross-Origin Resource Sharing (CORS)
- Specifying Server-Side Encryption
- Specifying Server-Side Encryption with Customer-Provided Encryption Keys

Amazon Simple Notification Service (Amazon SNS) Examples

The AWS SDK for .NET supports Amazon Simple Notification Service (Amazon SNS), which is a web service that enables applications, end users, and devices to instantly send and receive notifications from the cloud. For more information, see Amazon SNS.

The following example shows how to use the low-level APIs to list accessible topics in Amazon SNS. This example uses the default client constructor which constructs AmazonSimpleNotificationServiceClient with the credentials loaded from the application's default configuration, and if unsuccessful from the Instance Profile service on an EC2 instance.

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;

var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
  response = client.ListTopics(request);

  foreach (var topic in response.Topics)
  {
    Console.WriteLine("Topic: {0}", topic.TopicArn);

    var subs = client.ListSubscriptionsByTopic(
      new ListSubscriptionsByTopicRequest
      {
        TopicArn = topic.TopicArn
      });

    var ss = subs.Subscriptions;

    if (ss.Any())
    {
      Console.WriteLine("  Subscriptions:");

      foreach (var sub in ss)
      {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
      }
    }

    var attrs = client.GetTopicAttributes(
      new GetTopicAttributesRequest
      {
        TopicArn = topic.TopicArn
      }).Attributes;

    if (attrs.Any())
    {
      Console.WriteLine("  Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

} while (!string.IsNullOrEmpty(response.NextToken));
```

For related API reference information, see Amazon.SimpleNotificationService, Amazon.SimpleNotificationService.Model, and Amazon.SimpleNotificationService.Util in the *AWS SDK for .NET API Reference*.

Amazon SQS Examples

The AWS SDK for .NET supports Amazon SQS, which is a message queuing service that handles messages or workflows between components in a system. For more information, see Amazon SQS.

The following examples demonstrate how to use the AWS SDK for .NET to create and use Amazon SQS queues.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose **File**, **New Project**, **AWS Sample Projects**, **App Services**, **AWS SQS Sample**.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET.

For related API reference information, see Amazon.SQS, Amazon.SQS.Model, and Amazon.SQS.Util in the *AWS SDK for .NET Reference*.

Examples

Creating an Amazon SQS Client

You need an Amazon SQS client in order to create and use an Amazon SQS queue. Before configuring your client, you should create an App.Config file to specify your AWS credentials.

You specify your credentials by referencing the appropriate profile in the `appSettings` section of the file.

The following example specifies a profile named `my_profile`. For more information about credentials and profiles, see Configuring Your AWS SDK for .NET Application.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws profileName="my_profile"/>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

After you create this file, you're ready to create and initialize your Amazon SQS client.

1. Create and initialize an AmazonSQSConfig instance, and then set the `ServiceURL` property with the protocol and service endpoint, as follows.

   ```
   var sqsConfig = new AmazonSQSConfig();

   sqsConfig.ServiceURL = "http://sqs.us-west-2.amazonaws.com";
   ```

2. Use the `AmazonSQSConfig` instance to create and initialize an AmazonSQSClient instance, as follows.

   ```
   var sqsClient = new AmazonSQSClient(sqsConfig);
   ```

You can now use the client to create an Amazon SQS queue. For information about creating a queue, see Creating an Amazon SQS Queue.

Creating an Amazon SQS Queue

Creating an Amazon SQS queue is an administrative task that you can do by using the SQS Management Console. However, you can also use the AWS SDK for .NET to programmatically create an Amazon SQS queue.

1. Create and initialize a CreateQueueRequest instance. Provide the name of your queue and specify a visibility timeout for your queue messages, as follows.

   ```
   var createQueueRequest = new CreateQueueRequest();

   createQueueRequest.QueueName = "MySQSQueue";
   createQueueRequest.DefaultVisibilityTimeout = 10;
   ```

   Your queue name must be composed of only alphanumeric characters, hyphens, and underscores.

   Any message in the queue remains in the queue unless the specified visibility timeout is exceeded. The default visibility timeout for a queue is 30 seconds. For more information about visibility timeouts, see Visibility Timeout. For more information about different queue attributes you can set, see SetQueueAttributes.

2. After you create the request, pass it as a parameter to the CreateQueue method. The method returns a CreateQueueResponse object, as follows.

```
var createQueueResponse = sqsClient.CreateQueue(createQueueRequest);
```

For information about how queues work in Amazon SQS, see How SQS Queues Work.

For information about your queue URL, see Constructing Amazon SQS Queue URLs.

Constructing Amazon SQS Queue URLs

You need a queue URL to send, receive, and delete queue messages. A queue URL is constructed in the following format.

```
https://{REGION_ENDPOINT}/queue.|api-domain|/{YOUR_ACCOUNT_NUMBER}/{YOUR_QUEUE_NAME}
```

To find your AWS account number, go to Security Credentials. Your account number is located under **Account Number** at the top-right of the page.

For information about sending a message to a queue, see Sending an Amazon SQS Message.

For information about receiving messages from a queue, see Receiving a Message from an Amazon SQS Queue.

For information about deleting messages from a queue, see Deleting a Message from an Amazon SQS Queue.

Sending an Amazon SQS Message

You can use the AWS SDK for .NET to send a message to an Amazon SQS queue.

> **Important**
>
> Due to the distributed nature of the queue, Amazon SQS can't guarantee you will receive messages in the precise order they are sent. If you need to preserve the message order, place sequencing information in each message so you can reorder the messages upon receipt.

1. Create and initialize a SendMessageRequest instance. Specify the queue name and the message you want to send, as follows.

   ```
   sendMessageRequest.QueueUrl = myQueueURL; sendMessageRequest.MessageBody = "{YOUR_QUEUE
   ```

   For more information about your queue URL, see Constructing Amazon SQS Queue URLs.

   Each queue message must be composed of Unicode characters only, and can be up to 64 KB in size. For more information about queue messages, see SendMessage in the *Amazon SQS API Reference*.

2. After you create the request, pass it as a parameter to the SendMessage method. The method returns a SendMessageResponse object, as follows.

   ```
   var sendMessageResponse = sqsClient.SendMessage(sendMessageRequest);
   ```

The sent message will stay in your queue until the visibility timeout is exceeded, or until it is deleted from the queue. For more information about visibility timeouts, go to Visibility Timeout.

For information about deleting messages from your queue, see Deleting a Message from an Amazon SQS Queue.

For information about receiving messages from your queue, see Receiving a Message from an Amazon SQS Queue.

Sending an Amazon SQS Message Batch

You can use the AWS SDK for .NET to send batch messages to an Amazon SQS queue. The SendMessageBatch method delivers up to 10 messages to the specified queue. This is a batch version of SendMessage.

For a FIFO queue, multiple messages within a single batch are enqueued in the order they are sent.

For more information about sending batch messages, see SendMessageBatch in the *Amazon SQS API Reference*.

1. Create an AmazonSQSClient instance and initialize a SendMessageBatchRequest object. Specify the queue name and the message you want to send, as follows.

   ```
   AmazonSQSClient client = new AmazonSQSClient();
   var sendMessageBatchRequest = new SendMessageBatchRequest
   {
       Entries = new List<SendMessageBatchRequestEntry>
       {
           new SendMessageBatchRequestEntry("message1", "FirstMessageContent"),
           new SendMessageBatchRequestEntry("message2", "SecondMessageContent"),
           new SendMessageBatchRequestEntry("message3", "ThirdMessageContent")
       },
       QueueUrl = "SQS_QUEUE_URL"
   };
   ```

   For more information about your queue URL, see Constructing Amazon SQS Queue URLs.

   Each queue message must be composed of Unicode characters only, and can be up to 64 KB in size. For more information about queue messages, see SendMessage in the *Amazon SQS API Reference*.

2. After you create the request, pass it as a parameter to the SendMessageBatch method. The method returns a SendMessageBatchResponse object, which contains the unique ID of each message and the message content for each successfully sent message. It also returns the message ID, message content, and a sender's fault flag if the message failed to send.

   ```
   SendMessageBatchResponse response = client.SendMessageBatch(sendMessageBatchRequest);
   Console.WriteLine("Messages successfully sent:");
   foreach (var success in response.Successful)
   {
       Console.WriteLine("    Message id : {0}", success.MessageId);
       Console.WriteLine("    Message content MD5 : {0}", success.MD5OfMessageBody);
   }
   ```

```
Console.WriteLine("Messages failed to send:");
foreach (var failed in response.Failed)
{
    Console.WriteLine("    Message id : {0}", failed.Id);
    Console.WriteLine("    Message content : {0}", failed.Message);
    Console.WriteLine("    Sender's fault? : {0}", failed.SenderFault);
}
```

Receiving a Message from an Amazon SQS Queue

You can use the AWS SDK for .NET to receive messages from an Amazon SQS queue.

1. Create and initialize a ReceiveMessageRequest instance. Specify the queue URL to receive a message from, as follows.

   ```
   var receiveMessageRequest = new ReceiveMessageRequest();

   receiveMessageRequest.QueueUrl = myQueueURL;
   ```

   For more information about your queue URL, see Your Amazon SQS Queue URL.
2. Pass the request object as a parameter to the ReceiveMessage method, as follows.

   ```
   var receiveMessageResponse = sqsClient.ReceiveMessage(receiveMessageRequest);
   ```

   The method returns a ReceiveMessageResponse instance, containing the list of messages the queue contains.
3. The `ReceiveMessageResponse.ReceiveMessageResult` property contains a ReceiveMessageResponse object, which contains a list of the messages that were received. Iterate through this list and call the `ProcessMessage` method to process each message.

   ```
   foreach (var message in result.Messages)
   {
     ProcessMessage(message);   // Go to a method to process messages.
   }
   ```

   The `ProcessMessage` method can use the `ReceiptHandle` property to obtain a receipt handle for the message. You can use this receipt handle to change the message visibility timeout or to delete the message from the queue. For more information about how to change the visibility timeout for a message, see ChangeMessageVisibility.

For information about sending a message to your queue, see Sending an Amazon SQS Message.

For more information about deleting a message from the queue, see Deleting a Message from an Amazon SQS Queue.

Deleting a Message from an Amazon SQS Queue

You can use the AWS SDK for .NET to delete messages from an Amazon SQS queue.

1. Create and initialize a DeleteMessageRequest object. Specify the Amazon SQS queue to delete a message from and the receipt handle of the message to delete, as follows.

   ```
   var deleteMessageRequest = new DeleteMessageRequest();

   deleteMessageRequest.QueueUrl = queueUrl;
   deleteMessageRequest.ReceiptHandle = recieptHandle;
   ```

2. Pass the request object as a parameter to the DeleteMessage method. The method returns a DeleteMessageResponse object, as follows.

   ```
   var response = sqsClient.DeleteMessage(deleteMessageRequest);
   ```

   Calling `DeleteMessage` unconditionally removes the message from the queue, regardless of the visibility timeout setting. For more information about visibility timeouts, see Visibility Timeout.

For information about sending a message to a queue, see Sending an Amazon SQS Message.

For information about receiving messages from a queue, see Receiving a Message from an Amazon SQS Queue.

Enabling Long Polling in Amazon SQS

Long polling reduces the number of empty responses by allowing Amazon SQS to wait a specified time for a message to become available in the queue before sending a response. Also, long polling eliminates false empty responses by querying all the servers instead of a sampling of servers. To enable long polling, you must specify a non-zero wait time for received messages. You can do this by setting the `ReceiveMessageWaitTimeSeconds` parameter of a queue or by setting the `WaitTimeSeconds` parameter on a message when it's received. This .NET example shows you how to enable long polling in Amazon SQS for a newly created or existing queue, or upon receipt of a message.

These examples use the following methods of the AmazonSQSClient class to enable long polling:

- CreateQueue
- SetQueueAttributes
- ReceiveMessage

For more information about long polling, see Amazon SQS Long Polling in the *Amazon SQS Developer Guide*.

Enable Long Polling When Creating a Queue

Create an AmazonSQSClient service object. Create a CreateQueueRequest object containing the properties needed to create a queue, including a non-zero value for the `ReceiveMessageWaitTimeSeconds` property.

Call the CreateQueue method. Long polling is then enabled for the queue.

```
AmazonSQSClient client = new AmazonSQSClient();
var request = new CreateQueueRequest
{
    QueueName = "SQS_QUEUE_NAME",
    Attributes = new Dictionary<string, string>
```

```
    {
        { "ReceiveMessageWaitTimeSeconds", "20"}
    }
};
var response = client.CreateQueue(request);
Console.WriteLine("Created a queue with URL : {0}", response.QueueUrl);
```

Enable Long Polling on an Existing Queue

Create an AmazonSQSClient service object. Create a SetQueueAttributesRequest object containing the properties needed to set the attributes of the queue, including a non-zero value for the `ReceiveMessageWaitTimeSeconds` property and the URL of the queue. Call the SetQueueAttributes method. Long polling is then enabled for the queue.

```
AmazonSQSClient client = new AmazonSQSClient();

var request = new SetQueueAttributesRequest
{
    Attributes = new Dictionary<string, string>
    {
        { "ReceiveMessageWaitTimeSeconds", "20"}
    },
    QueueUrl = "SQS_QUEUE_URL"
};

var response = client.SetQueueAttributes(request);
```

Receive a Message

Create an AmazonSQSClient service object. Create a ReceiveMessageRequest object containing the properties needed to receive a message, including a non-zero value for the `WaitTimeSeconds` parameter and the URL of the queue. Call the ReceiveMessage method.

```
public void OnMessageReceipt()
{
    AmazonSQSClient client = new AmazonSQSClient();

    var request = new ReceiveMessageRequest
    {
        AttributeNames = { "SentTimestamp" },
        MaxNumberOfMessages = 1,
        MessageAttributeNames = { "All" },
        QueueUrl = "SQS_QUEUE_URL",
        WaitTimeSeconds = 20
    };

    var response = client.ReceiveMessage(request);
}
```

Using Amazon SQS Queues

Amazon SQS offers *standard* as the default queue type. A standard queue enables you to have a nearly-unlimited number of transactions per second. Standard queues support at-least-once message delivery. However, occasionally more than one copy of a message might be delivered out of order. Standard queues provide best-effort ordering, which ensures that messages are generally delivered in the same order as they're sent.

You can use standard message queues in many scenarios, as long as your application can process messages that arrive more than once and out of order.

This code example demonstrates how to use queues by using these methods of the :code:`AmazonSQSClient`class:

- ListQueues: Gets a list of your message queues
- GetQueueUrl: Obtains the URL for a particular queue
- DeleteQueue: Deletes a queue

For more information about Amazon SQS messages, see How Amazon SQS Queues Work in the *Amazon SQS Developer Guide*.

List Your Queues

Create a ListQueuesRequest object containing the properties needed to list your queues, which by default is an empty object. Call the ListQueues method with the `ListQueuesRequest` as a parameter to retrieve the list of queues. The ListQueuesResponse returned by the call contains the URLs of all queues.

```
AmazonSQSClient client = new AmazonSQSClient();

ListQueuesResponse response = client.ListQueues(new ListQueuesRequest());
foreach (var queueUrl in response.QueueUrls)
{
    Console.WriteLine(queueUrl);
}
```

Get the URL for a Queue

Create a GetQueueUrlRequest object containing the properties needed to identify your queue, which must include the name of the queue whose URL you want. Call the GetQueueUrl method using the `GetQueueUrlRequest` object as a parameter. The call returns a GetQueueUrlResponse object containing the URL of the specified queue.

```
AmazonSQSClient client = new AmazonSQSClient();

var request = new GetQueueUrlRequest
{
    QueueName = "SQS_QUEUE_NAME"
};

GetQueueUrlResponse response = client.GetQueueUrl(request);
Console.WriteLine("The SQS queue's URL is {1}", response.QueueUrl);
```

Delete a Queue

Create a DeleteQueueRequest object containing the URL of the queue you want to delete. Call the DeleteQueue method with the `DeleteQueueRequest` object as the parameter.

```
AmazonSQSClient client = new AmazonSQSClient();

var request = new DeleteQueueRequest
{
    QueueUrl = "SQS_QUEUE_URL"
};

client.DeleteQueue(request);
```

Using Amazon SQS Dead Letter Queues

This example shows you how to use a queue to receive and hold messages from other queues that the queues can't process.

A dead letter queue is one that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead letter queue to determine why their processing did not succeed. You must individually configure each source queue that sends messages to a dead letter queue. Multiple queues can target a single dead letter queue.

In this example, an AmazonSQSClient object uses the SetQueueAttributesRequest method to configure a source queue to use a dead letter queue.

For more information about Amazon SQS dead letter queues, see Using Amazon SQS Dead Letter Queues in the *Amazon SQS Developer Guide*.

Configure a Source Queue

This code example assumes you have created a queue to act as a dead letter queue. See Creating an Amazon SQS Queue for information about creating a queue. After creating the dead letter queue, you must configure the other queues to route unprocessed messages to the dead letter queue. To do this, specify a redrive policy that identifies the queue to use as a dead letter queue and the maximum number of receives by individual messages before they are routed to the dead letter queue.

Create an AmazonSQSClient object to set the queue attrributes. Create a SetQueueAttributesRequest object containing the properties needed to update queue attributes, including the `RedrivePolicy` property that specifies both the ARN of the dead letter queue, and the value of `maxReceiveCount`. Also specify the URL source queue you want to configure. Call the SetQueueAttributes method.

```
AmazonSQSClient client = new AmazonSQSClient();

var setQueueAttributeRequest = new SetQueueAttributesRequest
{
    Attributes = new Dictionary<string, string>
    {
        {"RedrivePolicy",   @"{ ""deadLetterTargetArn"" : ""DEAD_LETTER_QUEUE_ARN"", ""max
    },
    QueueUrl = "SOURCE_QUEUE_URL"
};
```

```
client.SetQueueAttributes(setQueueAttributeRequest)
```

Amazon CloudWatch Examples

Amazon CloudWatch is a web service that monitors your AWS resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. CloudWatch alarms send notifications or automatically make changes to the resources you're monitoring based on rules that you define.

The code for these examples is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio, a set of C# project templates are installed. The simplest way to start this project is to open Visual Studio, and then choose **File**, **New Project**, **AWS Sample Projects**, **Deployment and Management**, **AWS CloudWatch Example**.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET.

Examples

Describing, Creating, and Deleting Alarms in Amazon CloudWatch

This .NET example show you how to:

- Describe a CloudWatch alarm
- Create a CloudWatch alarm based on a metric
- Delete a CloudWatch alarm

The Scenario

An alarm watches a single metric over a time period you specify. It performs one or more actions based on the value of the metric, relative to a given threshold over a number of time periods. The following examples show how to describe, create, and delete alarms in CloudWatch using these methods of the `AmazonCloudWatchClient` class:

- DescribeAlarms
- PutMetricAlarm
- DeleteAlarms

For more information about CloudWatch alarms, see Creating Amazon CloudWatch Alarms in the *CloudWatch User Guide*.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Describing an Alarm

Create an AmazonCloudWatchClient instance and a DescribeAlarmsRequest object, limiting the alarms that are returned to those with a state of INSUFFICIENT_DATA. Then call the DescribeAlarms method of the `AmazonCloudWatchClient` object.

```
using (var cloudWatch = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var request = new DescribeAlarmsRequest();
    request.StateValue = "INSUFFICIENT_DATA";
    request.AlarmNames = new List<string> { "Alarm1", "Alarm2" };
    do
    {
        var response = cloudWatch.DescribeAlarms(request);
        foreach(var alarm in response.MetricAlarms)
        {
            Console.WriteLine(alarm.AlarmName);
        }
        request.NextToken = response.NextToken;
    } while (request.NextToken != null);
}
```

Creating an Alarm Based on a Metric

Create an AmazonCloudWatchClient instance and a PutMetricAlarmRequest object for the parameters needed to create an alarm that is based on a metric, in this case, the CPU utilization of an Amazon EC2 instance.

The remaining parameters are set to trigger the alarm when the metric exceeds a threshold of 70 percent.

Then call the PutMetricAlarm method of the `AmazonCloudWatchClient` object.

```
var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2);
client.PutMetricAlarm(new PutMetricAlarmRequest
{
    AlarmName = "Web_Server_CPU_Utilization",
    ComparisonOperator = ComparisonOperator.GreaterThanThreshold,
    EvaluationPeriods = 1,
    MetricName = "CPUUtilization",
    Namespace = "AWS/EC2",
    Period = 60,
    Statistic = Statistic.Average,
    Threshold = 70.0,
    ActionsEnabled = true,
    AlarmActions = new List<string> { "arn:aws:swf:us-west-2:" + "customerAccount" +
           ":action/actions/AWS_EC2.InstanceId.Reboot/1.0" },
    AlarmDescription = "Alarm when server CPU exceeds 70%",
    Dimensions = new List<Dimension>
        {
            new Dimension { Name = "InstanceId", Value = "INSTANCE_ID" }
        },
    Unit = StandardUnit.Seconds
};
```

Deleting an Alarm

Create an AmazonCloudWatchClient instance and a DeleteAlarmsRequest object to hold the names of the alarms you want to delete. Then call the DeleteAlarms method of the `AmazonCloudWatchClient` object.

```
using (var cloudWatch = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var response = cloudWatch.DeleteAlarms(
        new DeleteAlarmsRequest
        {
            AlarmNames = new List<string> { "Alarm1", "Alarm2" };
        });
}
```

Using Alarms in Amazon CloudWatch

This .NET example shows you how to change the state of your Amazon EC2 instances automatically based on a CloudWatch alarm.

The Scenario

Using alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your Amazon EC2 instances. You can use the stop or terminate actions when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances.

In this example, .NET is used to define an alarm action in CloudWatch that triggers the reboot of an Amazon EC2 instance. The methods use the AWS SDK for .NET to manage Amazon EC2 instances using these methods of the AmazonCloudWatchClient class:

- EnableAlarmActions
- DisableAlarmActions

For more information about CloudWatch alarm actions, see Create Alarms to Stop, Terminate, Reboot, or Recover an Instance in the *CloudWatch User Guide*.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Create and Enable Actions on an Alarm

1. Create an AmazonCloudWatchClient instance and a PutMetricAlarmRequest object to hold the parameters for creating an alarm, specifying `ActionsEnabled` as true and an array of ARNs for the actions the alarm will trigger. Call the PutMetricAlarm method of the `AmazonCloudWatchClient` object, which creates the alarm if it doesn't exist or updates it if the alarm does exist.

```
using (var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    client.PutMetricAlarm(new PutMetricAlarmRequest
    {
        AlarmName = "Web_Server_CPU_Utilization",
        ComparisonOperator = ComparisonOperator.GreaterThanThreshold,
        EvaluationPeriods = 1,
        MetricName = "CPUUtilization",
        Namespace = "AWS/EC2",
        Period = 60,
        Statistic = Statistic.Average,
        Threshold = 70.0,
        ActionsEnabled = true,
        AlarmActions = new List<string> { "arn:aws:swf:us-west-2:" + "customerAccount
        AlarmDescription = "Alarm when server CPU exceeds 70%",
        Dimensions = new List<Dimension>
        {
            new Dimension { Name = "InstanceId", Value = "instanceId" }
        },
        Unit = StandardUnit.Seconds
    });
}
```

2. When `PutMetricAlarm` completes successfully, create an EnableAlarmActionsRequest object containing the name of the CloudWatch alarm. Call the EnableAlarmActions method to enable the alarm action.

```
client.EnableAlarmActions(new EnableAlarmActionsRequest
{
    AlarmNames = new List<string> { "Web_Server_CPU_Utilization" }
});
```

3. Create a MetricDatum object containing the CPUUtilization custom metric. Create a PutMetricDataRequest object containing the `MetricData` parameter needed to submit a data point for the CPUUtilization metric. Call the PutMetricData method.

```
MetricDatum metricDatum = new MetricDatum
{ MetricName = "CPUUtilization" };
PutMetricDataRequest putMetricDatarequest = new PutMetricDataRequest
{
    MetricData = new List<MetricDatum> { metricDatum }
};
client.PutMetricData(putMetricDatarequest);
```

Disable Actions on an Alarm

Create an AmazonCloudWatchClient instance and a DisableAlarmActionsRequest object containing the name of the CloudWatch alarm. Call the DisableAlarmActions method to disable the actions for this alarm.

```
using (var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    client.DisableAlarmActions(new DisableAlarmActionsRequest
    {
        AlarmNames = new List<string> { "Web_Server_CPU_Utilization" }
    });
}
```

Getting Metrics from Amazon CloudWatch

This example shows you how to:

- Retrieve a list of CloudWatch metrics
- Publish CloudWatch custom metrics

The Scenario

Metrics are data about the performance of your systems. You can enable detailed monitoring of some resources such as your Amazon EC2 instances or your own application metrics. In this example, you use .NET to retrieve a list of published CloudWatch metrics and publish data points to CloudWatch metrics using these methods of the AmazonCloudWatchClient class:

- ListMetrics
- PutMetricData

For more information about CloudWatch metrics, see Using Amazon CloudWatch Metrics in the *CloudWatch User Guide*.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

List Metrics

Create a ListMetricsRequest object containing the parameters needed to list metrics within the `AWS/Logs` namespace. Call the ListMetrics method from a AmazonCloudWatchClient instance to list the `IncomingLogEvents` metric.

```
var logGroupName = "LogGroupName";
DimensionFilter dimensionFilter = new DimensionFilter()
{
    Name = logGroupName
};
var dimensionFilterList = new List<DimensionFilter>();
dimensionFilterList.Add(dimensionFilter);

var dimension = new Dimension
{
    Name = "UniquePages",
    Value = "URLs"
};
using (var cw = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var listMetricsResponse = cw.ListMetrics(new ListMetricsRequest
    {
        Dimensions = dimensionFilterList,
        MetricName = "IncomingLogEvents",
        Namespace = "AWS/Logs"
    });
    Console.WriteLine(listMetricsResponse.Metrics);
}
```

Submit Custom Metrics

Create a PutMetricDataRequest object containing the parameters needed to submit a data point for the `PAGES_VISITED` custom metric. Call the PutMetricData method from the AmazonCloudWatchClient instance.

```
using (var cw = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    cw.PutMetricData(new PutMetricDataRequest
    {
        MetricData = new List<MetricDatum>{new MetricDatum
        {
            MetricName = "PagesVisited",
            Dimensions = new List<Dimension>{dimension},
            Unit = "None",
            Value = 1.0
        }},
        Namespace = "SITE/TRAFFIC"
    });
}
```

Sending Events to Amazon CloudWatch Events

This .NET code example shows you how to:

- Create and update a scheduled rule to trigger an event
- Add a AWS Lambda function target to respond to an event
- Send events that are matched to targets

The Scenario

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources to various targets. Using simple rules, you can match events and route them to one or more target functions or streams. This .NET example shows you how to create and update a rule used to trigger an event, define one or more targets to respond to an event, and send events that are matched to targets for handling.

The code manages instances using these methods of the AmazonCloudWatchEventsClient class:

- PutRule
- PutTargets
- PutEvents

For more information about Amazon CloudWatch Events, see Adding Events with PutEvents in the *CloudWatch Events User Guide*.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.
- Create a Lambda function using the hello-world blueprint to serve as the target for events. To learn how, see Step 1: Create an AWS Lambda function in the *CloudWatch Events User Guide*.

Create an IAM Role to Run the Examples

The following examples require an IAM role whose policy grants permission to CloudWatch Events and that includes `events.amazonaws.com` as a trusted entity. This example creates a role named CWEvents, setting it's trust relationship and role policy.

```
static void Main()
{
    var client = new AmazonIdentityManagementServiceClient();
    // Create a role and it's trust relationship policy
    var role = client.CreateRole(new CreateRoleRequest
    {
        RoleName = "CWEvents",
        AssumeRolePolicyDocument =
        @"{""Statement"":[{""Principal"":{""Service"":[""events.amazonaws.com""]}," +
        @"""Effect"":""Allow"",""Action"":[""sts:AssumeRole""]}]}"
    }).Role;
    // Create a role policy and add it to the role
    string policy = GenerateRolePolicyDocument();
    var request = new CreatePolicyRequest
    {
        PolicyName = "DemoCWPermissions",
        PolicyDocument = policy
    };
    try
    {
        var createPolicyResponse = client.CreatePolicy(request);
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine
          ("Policy 'DemoCWPermissions' already exits.");
    }
    var request2 = new AttachRolePolicyRequest()
```

```
        {
            PolicyArn = "arn:aws:iam::192484417122:policy/DemoCWPermissions",
            RoleName = "CWEvents"
        };
        try
        {
            var response = client.AttachRolePolicy(request2);     //managedpolicy
            Console.WriteLine("Policy DemoCWPermissions attached to Role TestUser");
        }
        catch (NoSuchEntityException)
        {
            Console.WriteLine
              ("Policy 'DemoCWPermissions' does not exist");
        }
        catch (InvalidInputException)
        {
            Console.WriteLine
              ("One of the parameters is incorrect");
        }

    }
    public static string GenerateRolePolicyDocument()
    {
        /* This method produces the following managed policy:
       "Version": "2012-10-17",
       "Statement": [
         {
           "Sid": "CloudWatchEventsFullAccess",
           "Effect": "Allow",
           "Action": "events:*",
           "Resource": "*"
         },
         {
           "Sid": "IAMPassRoleForCloudWatchEvents",
           "Effect": "Allow",
           "Action": "iam:PassRole",
           "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
         }
       ]
     }
     */
        var actionList = new ActionIdentifier("events:*");
        var actions = new List<ActionIdentifier>();
        actions.Add(actionList);
        var resource = new Resource("*");
        var resources = new List<Resource>();
        resources.Add(resource);
        var statement = new Amazon.Auth.AccessControlPolicy.Statement
            (Amazon.Auth.AccessControlPolicy.Statement.StatementEffect.Allow)
        {
            Actions = actions,
            Id = "CloudWatchEventsFullAccess",
            Resources = resources
        };
        var statements = new List<Amazon.Auth.AccessControlPolicy.Statement>();
        statements.Add(statement);
        var actionList2 = new ActionIdentifier("iam:PassRole");
        var actions2 = new List<ActionIdentifier>();
        actions2.Add(actionList2);
        var resource2 = new Resource("arn:aws:iam::*:role/AWS_Events_Invoke_Targets");
        var resources2 = new List<Resource>();
        resources2.Add(resource2);
        var statement2 = new Amazon.Auth.AccessControlPolicy.Statement(Amazon.Auth.AccessContro
        {
            Actions = actions2,
            Id = "IAMPassRoleForCloudWatchEvents",
            Resources = resources2
        };

        statements.Add(statement2);
        var policy = new Policy
        {
            Id = "DemoEC2Permissions",
            Version = "2012-10-17",
            Statements = statements
        };
        return policy.ToJson();
    }
```

Create a Scheduled Rule

Create an AmazonCloudWatchEventsClient instance and a PutRuleRequest object containing the parameters needed to specify the new scheduled rule, which include the following:

- A name for the rule
- The ARN of the IAM role you created previously
- An expression to schedule triggering of the rule every five minutes

Call the PutRule method to create the rule. The PutRuleResponse returns the ARN of the new or updated rule.

```
AmazonCloudWatchEventsClient client = new AmazonCloudWatchEventsClient();

var putRuleRequest = new PutRuleRequest
{
    Name = "DEMO_EVENT",
    RoleArn = "IAM_ROLE_ARN",
    ScheduleExpression = "rate(5 minutes)",
    State = RuleState.ENABLED
};
```

```
var putRuleResponse = client.PutRule(putRuleRequest);
Console.WriteLine("Successfully set the rule {0}", putRuleResponse.RuleArn);
```

Add a Lambda Function Target

Create an AmazonCloudWatchEventsClient instance and a PutTargetsRequest object containing the parameters needed to specify the rule to which you want to attach the target, including the ARN of the Lambda function you created. Call the PutTargets method of the `AmazonCloudWatchClient` instance.

```
AmazonCloudWatchEventsClient client = new AmazonCloudWatchEventsClient();

var putTargetRequest = new PutTargetsRequest
{
    Rule = "DEMO_EVENT",
    Targets =
    {
        new Target { Arn = "LAMBDA_FUNCTION_ARN", Id = "myCloudWatchEventsTarget"}
    }
};
client.PutTargets(putTargetRequest);
```

Send Events

Create an AmazonCloudWatchEventsClient instance and a PutEventsRequest object containing the parameters needed to send events. For each event, include the source of the event, the ARNs of any resources affected by the event, and details for the event. Call the PutEvents method of the `AmazonCloudWatchClient` instance.

```
AmazonCloudWatchEventsClient client = new AmazonCloudWatchEventsClient();

var putEventsRequest = new PutEventsRequest
{
    Entries = new List<PutEventsRequestEntry>
    {
        new PutEventsRequestEntry
        {
            Detail = @"{ ""key1"" : ""value1"", ""key2"" : ""value2"" }",
            DetailType = "appRequestSubmitted",
            Resources =
            {
                "RESOURCE_ARN"
            },
            Source = "com.compnay.myapp"
        }
    }
};
client.PutEvents(putEventsRequest);
```

Using Subscription Filters in Amazon CloudWatch Logs

This .NET example shows you how to:

- List a subscription filter in CloudWatch Logs
- Create or delete a subscription filter in CloudWatch Logs

The Scenario

Subscriptions provide access to a real-time feed of log events from CloudWatch Logs and deliver that feed to other services such as an Amazon Kinesis Streams or AWS Lambda for custom processing, analysis, or loading to other systems. A subscription filter defines the pattern to use for filtering which log events are delivered to your AWS resource. This example shows how to list, create, and delete a subscription filter in CloudWatch Logs. The destination for the log events is a Lambda function.

This example uses the AWS SDK for .NET to manage subscription filters using these methods of the AmazonCloudWatchLogsClient class:

- DescribeSubscriptionFilters
- PutSubscriptionFilter
- DeleteSubscriptionFilter

For more information about CloudWatch Logs subscriptions, see Real-time Processing of Log Data with Subscriptions in the *CloudWatch Logs User Guide*.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Describe Existing Subscription Filters

Create an AmazonCloudWatchLogsClient object. Create a DescribeSubscriptionFiltersRequest object containing the parameters needed to describe your existing filters. Include the name of the log group and the maximum number of filters you want described. Call the DescribeSubscriptionFilters method.

```
public static void DescribeSubscriptionFilters()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.DescribeSubscriptionFiltersRequest()
    {
        LogGroupName = "GROUP_NAME",
        Limit = 5
    };
    try
    {
        var response = client.DescribeSubscriptionFilters(request);
    }
    catch (Amazon.CloudWatchLogs.Model.ResourceNotFoundException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Create a Subscription Filter

Create an AmazonCloudWatchLogsClient object. Create a PutSubscriptionFilterRequest object containing the parameters needed to create a filter, including the ARN of the destination Lambda function, the name of the filter, the string pattern for filtering, and the name of the log group. Call the PutSubscriptionFilter method.

```
public static void PutSubscriptionFilters()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.PutSubscriptionFilterRequest()
    {
        DestinationArn = "LAMBDA_FUNCTION_ARN",
        FilterName = "FILTER_NAME",
        FilterPattern = "ERROR",
        LogGroupName = "Log_Group"
    };
    try
    {
        var response = client.PutSubscriptionFilter(request);
    }
    catch (InvalidParameterException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Delete a Subscription Filter

Create an AmazonCloudWatchLogsClient object. Create a DeleteSubscriptionFilterRequest object containing the parameters needed to delete a filter, including the names of the filter and the log group. Call the DeleteSubscriptionFilter method.

```
public static void DeleteSubscriptionFilter()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.DeleteSubscriptionFilterRequest()
    {
        LogGroupName = "GROUP_NAME",
        FilterName = "FILTER"
    };
    try
    {
        var response = client.DeleteSubscriptionFilter(request);
    }
    catch (Amazon.CloudWatchLogs.Model.ResourceNotFoundException e)
```

```
        {
            Console.WriteLine(e.Message);
        }
}
```

AWS OpsWorks Programming with the AWS SDK for .NET

The AWS SDK for .NET supports AWS OpsWorks, which provides a simple and flexible way to create and manage stacks and applications. With AWS OpsWorks, you can provision AWS resources, manage their configuration, deploy applications to those resources, and monitor their health. For more information, see AWS OpsWorks.

The SDK provides APIs for programming with AWS OpsWorks. These APIs typically consist of sets of matching request-and-response objects that correspond to HTTP-based API calls focusing on their corresponding service-level constructs.

For related API reference information, see `Amazon.OpsWorks` and `Amazon.OpsWorks.Model` in the *AWS SDK for .NET Reference*.

Programming Additional AWS Services with the AWS SDK for .NET

The AWS SDK for .NET supports programming AWS services in addition to the ones that are described previously in this chapter. For information about programming specific services with the AWS SDK for .NET, see the *AWS SDK for .NET API Reference*.

In addition to the namespaces for individual AWS services, the AWS SDK for .NET also provides the following APIs:

Other general programming information for the AWS SDK for .NET includes the following:

- Overriding Endpoints in the AWS SDK for .NET
- .NET Object Lifecycles

Additional Resources

Home Page for AWS SDK for .NET

For more information about the AWS SDK for .NET, go to the home page for the SDK at http://aws.amazon.com/sdk-for-net/.

SDK Reference Documentation

The SDK reference documentation includes the ability to browse and search across all code included with the SDK. It provides thorough documentation, usage examples, and even the ability to browse method source. For more information, see the *AWS SDK for .NET Reference*.

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. Each documentation page has a **Go to the forums** button at the top of the page that takes you to the associated forum. AWS engineers monitor the forums and respond to questions, feedback, and issues. You can also subscribe to RSS feeds for any of the forums.

AWS Toolkit for Visual Studio

If you use the Microsoft Visual Studio IDE, you should check out the *Toolkit for Visual Studio User Guide*.

Document History

The following table describes the important changes since the last release of the *AWS SDK for .NET Developer Guide*.

Last major documentation update: July 28th, 2015

Change | Description	Release Date
New SDK version | Version 3 of the AWS SDK for .NET released.	July 28th, 2015

About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model: you are charged only for the services that you—or your applications—use. For new AWS users, a free usage tier is available. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier, see Use the AWS Free Tier. To obtain an AWS account, visit the AWS home page and click **Create a Free Account**.