AWS SDK for .NET Developer Guide

Release 1.0

Amazon Web Services

Dec 02, 2016

Contents

CHAPTER 1

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 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.

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. For more information about the AWS Tools for Windows PowerShell, see the Tools for Windows PowerShell User Guide.
- The AWS Toolkit for Visual Studio. For more information about 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 the AWSSDK assembly for a specific application project. For more information, see *Install AWS Assemblies with NuGet*.

Note: We recommend using Visual Studio Professional 2010 or higher to implement your applications. It is possible to use Visual Studio Express to implement applications with the the SDK, including installing the Toolkit for Visual Studio. However, the installation includes only the AWS project templates and the Standalone Deployment Tool. In particular, Toolkit for Visual Studio on Visual Studio Express does not support AWS Explorer.

1.1 How to Use This Guide

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

Getting Started with the AWS SDK for .NET How to install and configure the the SDK. If you have not used the the SDK 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 the SDK that applies to all AWS services. This chapter also includes information about how to migrate code to the latest version of the the SDK, and describes the differences between the last version and this one.

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

Additional Resources Additional resources outside of this guide that provide more information about AWS and the the SDK.

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

1.2 Supported Services and Revision History

The AWS SDK for .NET supports most AWS infrastructure products, and we regularly release updates to the the SDK to support new services and new service features. To see what changed with a given release, see the the SDK README file.

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

1.3 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. Also, to make AWS useful as a platform for prototyping and experimentation, AWS offers a free usage tier, in which services are free below a certain level of usage. For more information about AWS costs and the free usage tier go to Test-Driving AWS in the Free Usage Tier.

To obtain an AWS account, go to the AWS home page and click *Sign Up Now*.

CHAPTER 2

Getting Started with the AWS SDK for .NET

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

Tasks

- *Create an AWS Account and Credentials*
- *Install the .NET Development Environment*
- *Install the AWS SDK for .NET*
- *Start a New Project*

2.1 Create an AWS Account and Credentials

To access AWS, you need an AWS account.

To sign up for an AWS account

1. Open http://aws.amazon.com/, and then choose *Create an AWS Account*.
2. Follow the instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to http://aws.amazon.com and clicking *My Account/Console*.

To use the the SDK, 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 the SDK. Instead, create one or more IAM users, and use those credentials. For applications that run on EC2 instances, you can use IAM roles to provide temporary credentials.

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 the SDK. 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.

2.2 Install the .NET Development Environment

To use the the SDK, you must have the following installed.

2.2.1 Requirements

- (Required) Microsoft .NET Framework 3.5 or later
- (Required) Microsoft Visual Studio 2010 or later
- (Required) The the SDK
- (Recommended) AWS Toolkit for Visual Studio, a plugin that provides a user interface for managing your AWS resources from Visual Studio, and includes the the SDK. For more information, see Using the AWS Toolkit for Visual Studio.

Note: We recommend using Visual Studio Professional 2010 or higher to implement your applications.

2.3 Install the AWS SDK for .NET

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

To install the the SDK

1. Go to AWS SDK for .NET. Click the *Download* button in the upper right corner of the page. Your browser will prompt you to save the install file.

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

   ---

2. To begin the install process, open the saved install file and follow the on-screen instructions. Version 2 of the the SDK can be found in the `past-releases` folder of the the SDK installation directory.

   Tip: By default, the AWS Tools for Windows is installed in the *Program Files* directory, which requires administrator privileges. To install the AWS Tools for Windows as a non-administrator,

specify a different installation directory.

3. (Optional) You can install extensions for the the SDK, which include a session state provider and a trace listener. For more information, see *Install AWS Assemblies with NuGet*.

2.4 Start a New Project

If you have installed the Toolkit for Visual Studio on Visual Studio Professional, it 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.

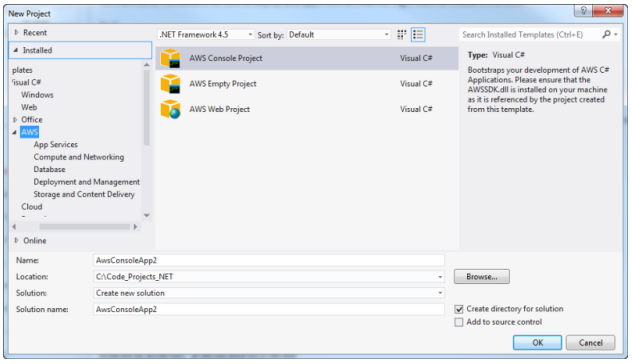
AWS Web Project An ASP.NET application that makes basic requests to Amazon S3, Amazon SimpleDB, and Amazon EC2.

You can also base your application on one of the standard Visual Studio project templates. Just add a reference to the AWS .NET library (`AWSSDK.dll`), which is located in the `past-releases` folder of the the SDK installation directory.

The following procedure gets you 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 on how to configure an AWS application, see *Configuring Your AWS SDK for .NET Application*.

To start a new project

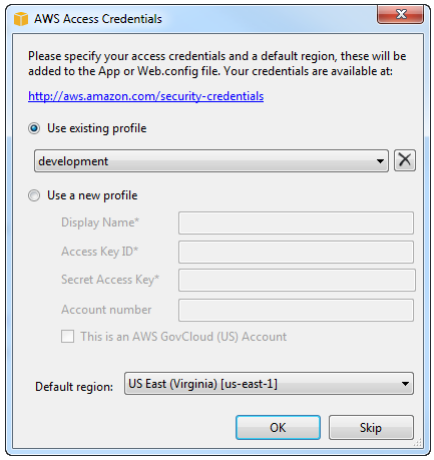
1. In Visual Studio, on the *File* menu, select *New*, and then click *Project* to open the *New Project* dialog box.
2. Select *AWS* from the list of installed templates and select the *AWS Console Project* project template. Enter a project name, and then click *OK*.



3. Use the *AWS Access Credentials* dialog box to configure your application.
   - Specify which account profile your code should use to access AWS. To use an existing profile, click *Use existing profile* and select the profile from the list. To add a new profile,

click *Use a new profile* and enter the credentials information. For more information about profiles, see *Configuring Your AWS SDK for .NET Application*.

- Specify a default AWS region.



4. Click *OK* to accept the configuration, which opens the project. Examine the project's `App.config` file, which will contain something like the following:

```
<configuration>
    <appSettings>
        <add key="AWSProfileName" value="development"/>
        <add key="AWSRegion" value="us-west-2"/>
    </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 that you move to using the `aws` element instead, for example:

```
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws region="us-west-2" profileName="development"/>
</configuration>
```

For more information on use of the `aws` element, see *Configuration Files Reference for AWS SDK for .NET*.

---

5. Click `F5` to compile and run the application, which prints the number of 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*.

CHAPTER 3

Programming with the AWS SDK for .NET

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

For information about developing software with the AWS SDK for .NET for specific AWS services, see *Programming AWS Services with the AWS SDK for .NET*.

3.1 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"/>
  </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*.

3.1.1 Configuring AWS Credentials

This topic describes how to configure your application's AWS credentials. It assumes you have created an AWS account and have access to your credentials, as described in *Create an AWS Account and Credentials*. It is important to manage your credentials securely and avoid practices that could unintentionally expose your credentials publicly. In particular:

- 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. Doing so creates a risk of accidentally exposing your credentials if, for example, you upload the project to a public repository.

Some general guidelines for securely managing credentials include:

- Create IAM users and use the credentials for the IAM users instead of your account's root credentials to provide account access. IAM user credentials are easier to revoke if they are compromised. You can apply to each IAM user a policy that restricts the user to a specified set of resources and actions.
- The preferred approach for managing credentials during application development is to put a profile for each set of IAM user credentials in the SDK Store. You can also use a credentials file to store profiles that contain credentials. You can then reference a particular profile programmatically or in your application's `App.config` or `Web.config` file instead of storing the credentials in your project files. To limit the risk of unintentionally exposing credentials, the SDK Store or credentials file should be stored separately from your project files.
- Use IAM roles for applications that are running on Amazon EC2 instances.
- Use temporary credentials 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 general discussion of how to securely manage AWS credentials, see Best Practices for Managing AWS Access Keys.

Topics

- *Using the SDK Store*
- *Using a Credentials File*

- *Using Credentials in an Application*

Using the SDK Store

During development of your AWS SDK for .NET application, you should add a profile to the SDK Store for each set of credentials you want to use in your application. This will prevent the accidental exposure of your AWS credentials while developing your application. 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, which limits the risk of accidentally exposing your credentials.
- You reference the profile by name in your application and the associated credentials are incorporated at build time. Your source files never contain the credentials.
- If you include a profile named `default`, the AWS SDK for .NET will use that profile by default.
- The SDK Store also provides credentials to the Tools for Windows PowerShell User Guide.

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, SDK Store profiles cannot be used in production applications. For more information, see *Using Credentials in an Application*.

There are several ways to manage the profiles in the SDK Store.

- The Toolkit for Visual Studio includes a graphical user interface for managing profiles. For more information about adding credentials to the SDK Store with the graphical user interface, see Specifying Credentials in the Toolkit for Visual Studio User Guide.
- You can manage your profiles from the command line by using the AWS Tools for Windows PowerShell. For more information, see Using AWS Credentials in the Tools for Windows PowerShell User Guide.
- You can manage your profiles programmatically using the Amazon.Util.ProfileManager class. The following example uses the RegisterProfile method to add a new profile to the SDK Store.

```
Amazon.Util.ProfileManager.RegisterProfile({profileName}, {accessKey},
↪{secretKey})
```

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

Using a Credentials File

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

You use a text editor to manage the profiles in a credentials file. 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`.

Each profile has the following format:

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

A profile can optionally include a session token. For more information, see Best Practices for Managing AWS Access Keys.

Tip: If you include a profile named `default`, the AWS SDK for .NET will use that profile by default if it cannot find the specified profile.

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

Using Credentials in an Application

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

1. Access key and secret key values that are stored in the application's `App.config` or `Web.config` file. We strongly recommend using profiles rather than storing literal credentials in your project files.
2. If a profile is specified:
   (a) The specified profile in the SDK Store.
   (b) The specified profile in the credentials file.

   If no profile is specified:
   (a) A profile named `default` in the SDK Store.
   (b) A profile named `default` in the credentials file.
3. Credentials stored in the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_KEY` environment variables.
4. For applications running on an Amazon EC2 instance, credentials stored in an instance profile.

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, SDK Store profiles cannot be used in production applications. If your application is running on an Amazon EC2 instance, you should use an IAM role as described in *Using IAM Roles for EC2 Instances with the AWS SDK for .NET*. Otherwise, you should store your credentials in a credentials file on the server your web application has access to.

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 recommended way to specify a profile is to define an `<aws>` element in 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>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
  </configSections>
  <aws profileName="development"/>
</configuration>
```

Note: The `<configSections>` element must be the first child of the `<configuration>` element.

Another 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 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 `profilesLocation` attribute value to the `<aws>` element. The following example specifies `C:aws_service_credentialscredentials` as the credentials file by using the recommended `<aws>` element.

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

Another way to specify a credentials file is with the `<appSettings>` element.

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

Although you can reference a profile programmatically using the Amazon.Runtime.StoredProfileAWSCredentials class, we recommend that you use the `aws` element instead. The following example demonstrates how to create an `AmazonS3Client` object that uses the credentials for a specific profile.

```
var credentials = new StoredProfileAWSCredentials(profileName);
var s3Client = new AmazonS3Client(credentials, RegionEndpoint.USWest2);
```

Tip: If you want to use the default profile, omit the `AWSCredentials` object, and the AWS SDK for .NET will automatically use your default credentials to create the client object.

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 will be 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, go to:

- 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 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 have been deprecated.

3.1.2 AWS Region Selection

AWS regions allow you to access AWS services that reside physically in a specific geographic region. This can be useful both for redundancy and to keep your data and applications running close to where you and your users will access them. To select a particular region, configure the AWS client object with an endpoint that corresponds to that region.

For example:

```
AmazonEC2Config config = new AmazonEC2Config();
config.ServiceURL = "https://us-east-1.amazonaws.com";
Amazon.Runtime.AWSCredentials credentials = new Amazon.Runtime.
 ↪StoredProfileAWSCredentials("profile_name");
AmazonEC2Client ec2 = new AmazonEC2Client(credentials, config);
```

You can also specify the region using the RegionEndpoint class. Here is an example that instantiates an Amazon EC2 client using AWSClientFactory and specifies the region:

```
Amazon.Runtime.AWSCredentials credentials = new Amazon.Runtime.
 ↪StoredProfileAWSCredentials("profile_name");
AmazonEC2Client ec2 = AWSClientFactory.CreateAmazonEC2Client(
   credentials, RegionEndpoint.USEast1 );
```

Regions are isolated from each other. For example, you can't access *US East* resources when using the *EU West* region. If your code needs access to multiple AWS regions, we recommend that you create a client specific to each region.

Regions are logically isolated from each other; you can't access another region's resources when communicating with the China (Beijing) Region endpoint.

Go to Regions and Endpoints in the Amazon Web Services General Reference to view the current list of regions and corresponding endpoints for each of the services offered by AWS.

3.1.3 Configuring Other Application Parameters

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

- AWSEndpointDefinition
- AWSLogging
- AWSLogMetrics
- AWSRegion
- AWSResponseLogging

- AWS.DynamoDBContext.TableNamePrefix
- AWS.S3.UseSignatureVersion4

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*.

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.xml"/>
```

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

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

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

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

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.

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"/>
```

lternatively, 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-"/>
```

```
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.

3.1.4 Configuration Files Reference for AWS SDK for .NET

You can use a .NET project's `App.config` or `Web.config` file to specify certain AWS settings such as AWS credentials, logging options, AWS service endopoints, and AWS regions, as well as certain 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 that 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*.)

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*.)

Topics

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, as shown in the following example:

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

Note that the Visual Studio Editor does *not* provide automatic code completion (IntelliSense) for either 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 that 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">
      <alias
        fromTable="string value"
        toTable="string value" />
      <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.

Topics

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 that is 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 *<dynamoDBContext>* 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 desired AWS regions and endpoints to use. (This attribute maps to the Amazon.AWSConfigs.EndpointDefinition property in the AWS SDK for .NET.)

profileName The desired 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 element.

The `<dynamoDB>` element can include the 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 that 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 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 to 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. (This attribute 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. (This attribute 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 Log service responses only when there are errors.

logTo Where to log to. (This attribute maps to the A:code:*mazon.Util.LoggingConfig.LogTo* property from the `Amazon.AWSConfigs.LoggingConfig.LogTo` property in the AWS SDK for .NET.)

Allowed values include:

Log4Net Log to log4net.

None Completely disable logging.

SystemDiagnostics Log to System.Diagnostics.

The parent of the `<logging>` element is the 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. (This element maps to an instance of the Amazon.Util.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 element.

The `<map>` element can include one or more instances of the 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 Amazon.Util.TypeMapping.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 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 the SDK 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 username to authenticate with the proxy server. (This attributes maps to the `Amazon.Util.ProxyConfig.Username` property from the `mazon.AWSConfigs.ProxyConfig.Username` property in the AWS SDK for .NET.)

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

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

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

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

3.2 Amazon Web Services Asynchronous APIs for .NET

Topics

- *Asynchronous API for .NET 4.5, Windows Store, and Windows Phone 8*
- *Asynchronous API for .NET 3.5*

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

The AWS SDK for .NET uses the new task-based asynchronous pattern for .NET 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.

3.2.2 Asynchronous API for .NET 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 could 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. Then, after some period of time, you would call the corresponding `End` method. This method retrieves the response from the service and also provides an opportunity to handle exceptions that might have occurred during the operation.

Note: It is not required that you call the `End` method. Assuming that 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 completes. 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.

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
  //
}
```

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`.

No Callback Specified

The following example code calls `BeginPutObject`, performs some work, 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 that 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 completes, 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 that 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 completes, 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 that 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 completes, 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 various patterns that 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*

3.3 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 the SDK 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?

Topics

3.3.1 Retries

The the SDK 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 the SDK 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.

3.3.2 Timeouts

The the SDK 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 net-api-v2:*ClientConfig <TRuntimeClientConfigNET45>* 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 the SDK that targets the portable class library (PCL) uses the HttpClient class instead of the `HttpWebRequest` class, and supports the Timeout property only.

- `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 the SDK 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 the SDK that targets the portable class library (PCL) sets `Timeout` to the maximum value for all AmazonS3Client and AmazonGlacierClient objects.

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 the SDK 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 the SDK that targets the portable class library (PCL) sets `Timeout` to the maximum value for all AmazonS3Client and AmazonGlacierClient objects.

3.3.3 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
  });
```

3.4 Migrating Your Code to the Version 2 of the AWS SDK for .NET

This guide describes changes in the version 2 of the SDK, and how you can migrate your code to this version of the SDK.

Topics

3.4.1 Introduction

The AWS SDK for .NET was released in November 2009 and was originally designed for .NET Framework 2.0. Since then, .NET has improved with .NET 4.0 and .NET 4.5. Since .NET 2.0, .NET has also added new target platforms: WinRT and Windows Phone 8.

AWS SDK for .NET version 2 has been updated to take advantage of the new features of the .NET platform and to target WinRT and Windows Phone 8.

3.4.2 What's New

- Support for `Task`-based asynchronous API
- Support for Windows Store apps
- Support for Windows Phone 8
- Ability to configure service region via `App.config` or `Web.config`
- Collapsed `Response` and `Result` classes
- Updated names for classes and properties to follow .NET conventions

3.4.3 What's Different

Architecture

The AWS SDK for .NET uses a common runtime library to make AWS service requests. In version 1 of the SDK, this "common" runtime was added *after the initial release*, and several of the older AWS services did not use it. As a result, there was a higher degree of variability among services in the functionality provided by the AWS SDK for .NET version 1.

In version 2 of the SDK, all services now use the common runtime, so future changes to the core runtime will propagate to all services, increasing their uniformity and easing demands on developers who want to target multiple services.

However, separate runtimes are provided for .NET 3.5 and .NET 4.5:

- The version 2 runtime for *.NET 3.5* is similar to the existing version 1 runtime, which is based on the System.Net.HttpWebRequest class and uses the `Begin` and `End` pattern for asynchronous methods.

- The version 2 runtime for *.NET 4.5* is based on the new System.Net.Http.HttpClient class and uses `Tasks` for asynchronous methods, which enables users to use the new `async` and `await` keywords in C# 5.0.

The WinRT and Windows Phone 8 versions of the SDK reuse the runtime for .NET 4.5, with the exception that they support *asynchronous methods* only. Windows Phone 8 doesn't natively support `System.Net.Http.HttpClient`, so the SDK depends on Microsoft's portable class implementation of `HttpClient`, which is hosted on *NuGet* at the following URL:

- http://nuget.org/packages/Microsoft.Net.Http/2.1.10

Removal of the "With" Methods

The "With" methods have been removed from version 2 of the SDK for the following reasons:

- In .NET 3.0, *constructor initializers* were added, making the "With" methods redundant.
- The "With" methods added significant overhead to the API design and worked poorly in cases of inheritance.

For example, in version 1 of the SDK, you would use "With" methods to set up a `TransferUtilityUploadRequest`:

```
TransferUtilityUploadRequest uploadRequest = new␣
↪TransferUtilityUploadRequest()
  .WithBucketName("my-bucket")
  .WithKey("test")
  .WithFilePath("c:\test.txt")
  .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256);
```

In the current version of the SDK, use constructor initializers instead:

```
TransferUtilityUploadRequest uploadRequest = new␣
↪TransferUtilityUploadRequest() {
  BucketName = "my-bucket", Key = "test", FilePath = "c:\test.txt",
  ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
};
```

Removal of SecureString

The use of `System.Security.SecureString` was removed in version 2 of the SDK because it is not available on the WinRT and Windows Phone 8 platforms.

Breaking Changes

Many classes and properties were changed to either meet .NET naming conventions or more closely follow service documentation. Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2) were the most affected by this because they are the oldest services in the SDK and were moved to the new common runtime. Below are the most visible changes.

- All client interfaces have been renamed to follow the .NET convention of starting with the letter "I". For example, the `AmazonEC2` class is now IAmazonEC2.
- Properties for collections have been properly pluralized.
- `AWSClientFactory.CreateAmazonSNSClient` has been renamed CreateAmazonSimpleNotificationServiceClient.
- `AWSClientFactory.CreateAmazonIdentityManagementClient` has been renamed CreateAmazonIdentityManagementServiceClient.

Amazon DynamoDB

- The `amazon.dynamodb` namespace has been removed; only the amazon.dynamodbv2 namespace remains.
- Service-response collections that were set to null in version 1 are now set to an empty collection. For example, QueryResult.LastEvaluatedKey and ScanResponse.LastEvaluatedKey will be set to *empty* collections when there are no more items to query/scan. If your code depends on `LastEvaluatedKey` to be `null`, it now has to check the collection's `Count` field to avoid a possible infinite loop.

Amazon EC2

- `Amazon.EC2.Model.RunningInstance` has been renamed Instance.

  Additionally, the `GroupName` and `GroupId` properties of `RunningInstance` have been combined into the SecurityGroups property, which takes a GroupIdentifier object, in `Instance`.
- `Amazon.EC2.Model.IpPermissionSpecification` has been renamed IpPermission.
- `Amazon.EC2.Model.Volume.Status` has been renamed State.
- AuthorizeSecurityGroupIngressRequest removed root properties for `ToPort` and `FromPort` in favor of always using IpPermissions.

  This was done because the root properties were silently ignored when set for an instance running in a VPC.
- The AmazonEC2Exception class is now based on AmazonServiceException instead of `System.Exception`.

  As a result, many of the exception properties have changed; the `XML` property is no longer provided, for example.

Amazon Redshift

- The `ClusterVersion.Name` property has been renamed ClusterVersion.Version.

Amazon S3

- `AmazonS3Config.CommunicationProtocol` was removed to be consistent with other services where ServiceURL contains the protocol.
- The `PutACLRequest.ACL` property has been renamed AccessControlList to make it consistent with GetACLResponse.
- `GetNotificationConfigurationRequest/Response` and `SetNotificationConfigurationRequest/Response` have been renamed GetBucketNotificationRequest/ Response and PutBucketNotificationRequest/ Response, respectively.
- `EnableBucketLoggingRequest/Response` and `DisableBucketLoggingRequest/Response` were consolidated into PutBucketLoggingRequest/ Response.
- The `GenerateMD5` property has been removed from PutObjectRequest and UploadPartRequest because this is now automatically computed as the object is being written to Amazon S3 and compared against the MD5 returned in the response from Amazon S3.
- The `PutBucketTagging.TagSets` collection is now PutBucketTagging.TagSet, and now takes a list of Tag objects.
- The AmazonS3Util utility methods DoesS3BucketExist, SetObjectStorageClass, SetServerSideEncryption, SetWebsiteRedirectLocation, and DeleteS3BucketWithObjects were changed to take IAmazonS3 as the first parameter to be consistent with other high-level APIs in the SDK.
- Only responses that return a `Stream` like GetObjectResponse are `IDisposable`. In version 1, all responses were `IDisposable`.
- The `BucketName` property has been removed from Amazon.S3.Model.S3Object.

Amazon Simple Workflow Service

- The `DomainInfos.Name` property has been renamed DomainInfos.Infos.

Configuring the AWS Region

Regions can be set in the `App.config` or `Web.config` files (depending on your project type). The recommended approach is to use the `aws` element, although using the `appSettings` element is still supported.

For example, the following specification configures all clients that don't explicitly set the region to point to **|region_api_default|** through use of the `aws` element.

```
<configuration> <configSections> <section name="aws" type="Amazon.AWSSection,␣
↪AWSSDK"/> </configSections> <aws profileName="{profile_name}" region=
↪"|region_console_default|"/>
</configuration>
```

Alternatively, you can use the `appSettings` element.

```
 <configuration> <appSettings> <add key="AWSProfileName" value="{profile_name}
↪"/>
    <add key="AWSRegion" value="|region_console_default|"/>
  </appSettings>
</configuration>
```

Response and Result Classes

To simplify your code, the `Response` and `Result` classes that are returned when creating a service object have been collapsed. For example, the code to get an Amazon SQS queue URL previously looked like this:

```
GetQueueUrlResponse response = SQSClient.GetQueueUrl(request);
Console.WriteLine(response.CreateQueueResult.QueueUrl);
```

You can now get the queue URL simply by referring to the `QueueUrl` member of the CreateQueueResponse returned by the AmazonSQSClient.CreateQueue method:

```
Console.WriteLine(response.QueueUrl);
```

The `CreateQueueResult` property still exists, but has been marked as *deprecated*, and may be removed in a future version of the SDK. Use the `QueueUrl` member instead.

Additionally, all of the service response values are based on a common response class, AmazonWebServiceResponse, instead of individual response classes per service. For example, the PutBucketResponse class in Amazon S3 is now based on this common class instead of `S3Response` in version 1. As a result, the methods and properties available for `PutBucketResponse` have changed.

Refer to the return value type of the *Create** method for the service client that you're using to see what values are returned. These are all listed in the AWS SDK for .NET Reference.

3.5 Platform Differences in the AWS SDK for .NET

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

3.5.1 AWS SDK for .NET Framework 3.5

This version of the the SDK is the one most similar to version 1. This version, compiled against .NET Framework 3.5, supports the same set of services as version 1. It also uses the same *pattern for making asynchronous calls*.

Note: This version contains a number of changes that may break code that was designed for version 1. For more information, see the *Migration Guide*.

3.5.2 AWS SDK for .NET Framework 4.5

The version of the the SDK compiled against .NET Framework 4.5 supports the same set of services as version 1 of the SDK. However, it uses a different pattern for asynchronous calls. Instead of the Begin/End pattern it uses the task-based pattern, which allows developers to use the new async and await keywords introduced in C# 5.0.

3.5.3 AWS SDK for Windows RT

The version of the the SDK compiled for WinRT supports only asynchronous method calls using `async` and `await`.

This version does not provide all of the functionality for Amazon S3 and DynamoDB that was available in version 1 of the SDK. The following Amazon S3 functionality is currently unavailable in the Windows RT version of SDK.

- Transfer Utility
- IO Namespace

The Windows RT version of the SDK does not support decryption of the Windows password using the GetDecryptedPassword method.

3.5.4 AWS SDK for Windows Phone 8

The version of the the SDK compiled for Windows Phone 8 has a programming model similar to Windows RT. As with the Windows RT version, it supports only asynchronous method calls using `async` and `await`. Also, because Windows Phone 8 doesn't natively support `System.Net.Http.HttpClient`, the SDK depends on Microsoft's portable class implementation of `HttpClient`, which is hosted on nuget at the following URL:

- http://nuget.org/packages/Microsoft.Net.Http/2.1.10

This version of the AWS SDK for .NET supports the same set of services supported in the AWS Mobile SDK for Android and the AWS Mobile SDK for iOS:

- Amazon EC2
- Elastic Load Balancing
- Auto Scaling
- Amazon S3
- Amazon SNS
- Amazon SQS

- Amazon SES
- DynamoDB
- Amazon SimpleDB
- CloudWatch
- AWS STS

This version does not provide all of the functionality for Amazon S3 and DynamoDB available in version 1 of the SDK. The following Amazon S3 functionality is currently unavailable in the Windows Phone 8 version of SDK.

- Transfer Utility
- IO Namespace

Also, the Windows Phone 8 version of the SDK does not support decryption of the Windows password using the GetDecryptedPassword method.

3.6 Install AWS Assemblies with NuGet

NuGet is a package management system for the .NET platform. With NuGet, you can add the AWSSDK assembly and the TraceListener and SessionProvider extensions to your application without first installing the SDK.

NuGet always has the most recent versions of the AWS .NET assemblies, and also enables you to install previous versions. NuGet is aware of dependencies between assemblies and installs required assemblies automatically. Assemblies that are installed with NuGet are stored with your solution rather than in a central location such as `Program Files`. This enables you to install assembly versions specific to a given application without creating compatibility issues for other applications.

For more information about NuGet, go to the NuGet documentation.

Topics

- *Installation*
- *NuGet from Solution Explorer*
- *NuGet Package Manager Console*

3.6.1 Installation

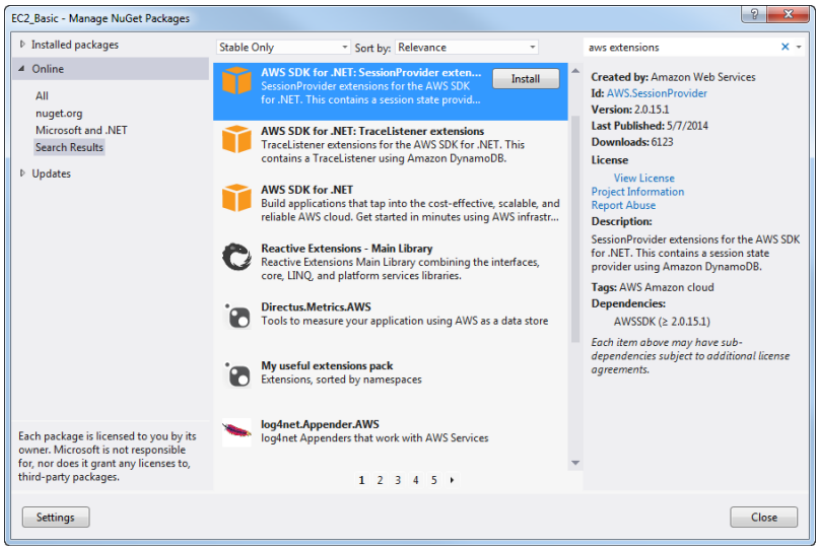
To use NuGet, install it from the Visual Studio Gallery on MSDN. If you are using Visual Studio 2010 or later, NuGet is installed automatically.

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

3.6.2 NuGet from Solution Explorer

To use NuGet from Solution Explorer, right-click on your project and select *Manage NuGet Packages...* from the context menu.

From the *Manage NuGet Packages* dialog box, select *Online* in the left pane. You can then search for the package that you want to install using the search box in the upper right corner. The screenshot shows the `AWS.Extensions` assembly package. Notice that NuGet is aware that this package has a dependency on the `AWSSDK` assembly package; NuGet will therefore install the `AWSSDK` package if it is not already installed.



3.6.3 NuGet Package Manager Console

To use NuGet from the Package Manager Console within Visual Studio:

- Visual Studio 2010 – From the *Tools* menu, select *Library Package Manager*, and click *Package Manager Console*.
- Visual Studio 2012 – From the *Tools* menu, select *Nuget Package Manager*, and click *Package Manager Console*.

From the console, you can install the AWS assemblies using the *Install-Package* command. For example, to install the AWS SDK for .NET assembly, use the following command line:

```
PM> Install-Package AWSSDK
```

To install an earlier version of a package, use the `-Version` option and specify the desired package version. For example, to install version 1.5.1.0 of the AWS SDK for .NET assembly, use the following command line:

```
PM> Install-Package AWSSDK -Version 1.5.1.0
```

The NuGet website provides a page for every package that is available through NuGet such as the AWSSDK and AWS.Extensions assemblies. The page for each package includes a sample command line

for installing the package using the console. Each page also includes a list of the previous versions of the package that are available through NuGet.

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

CHAPTER 4

Programming AWS Services with the AWS SDK for .NET

The following concepts, tutorials, and examples demonstrate how to use the AWS SDK for .NET to work with individual Amazon Web Services.

Before you begin, be sure that you have *set up the AWS SDK for .NET* and that you have reviewed the material in the *Programming with the AWS SDK for .NET*.

4.1 Programming with the AWS Resource APIs for .NET

The AWS SDK for .NET provides the AWS Resource APIs for .NET. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with resources that are managed by AWS services. A resource is a logical object that is exposed by an AWS service's APIs. For example, AWS Identity and Access Management (IAM) exposes users and groups as resources that can be programmatically accessed more directly by these resource APIs than by other means.

The AWS Resource APIs for .NET are currently provided as a preview. This means that these resource APIs may frequently change in response to customer feedback, and these changes may happen without advance notice. Until these resource APIs exit the preview stage, please be cautious about writing and distributing production-quality code that relies on them.

Using the AWS Resource APIs for .NET provide these benefits:

- The resource APIs in the the SDK are easier to understand conceptually than their low-level API counterparts. The low-level APIs in the the SDK typically consist of sets of matching request-and-response objects that correspond to HTTP-based API calls focusing on somewhat isolated AWS service constructs. In contrast, these resource APIs represent logical relationships among resources within AWS services and intuitively use familiar .NET programming constructs.
- Code that you write with the resource APIs is easier for you and others to comprehend when compared to their low-level API equivalents. Instead of writing somewhat complex request-and-response style code with the low-level APIs to access resources, you can get directly to resources with the resource APIs. If you're working with a team of developers in the same code base, it's typically easier to understand what has already been coded and to start contributing quickly to existing code.
- You will typically write less code with the resource APIs than with equivalent low-level API code. Request-and-response style code with the low-level APIs can sometimes be quite long. Equivalent

resource APIs code is typically much shorter, more compact, and easer to debug.

Here's a brief example of using C# and the AWS Resource APIs for .NET to create a new IAM user account:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var user = iam.CreateUser("DemoUser");

  Console.WriteLine("User Name = '{0}', ARN = '{1}'",
    user.Name, user.Arn);
}
catch (EntityAlreadyExistsException)
{
  Console.WriteLine("User 'DemoUser' already exists.");
}
```

Compare this to an equivalent example of using the low-level APIs:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

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.");
}
```

Even with this brief code example, you'll see that the resource APIs code is a bit easier to comprehend than the low-level code, and the resource APIs code is a bit shorter and more compact than its low-level counterpart.

There are a few limitations to note when using the resource APIs as compared to the low-level APIs in the the SDK:

- Not all of the AWS services currently have resource APIs (although this number is growing). Currently, the following AWS services have resource APIs in the the SDK:

  - AWS CloudFormation
  - Amazon Glacier
  - AWS Identity and Access Management (IAM)
  - Amazon Simple Notification Service (Amazon SNS)
  - Amazon Simple Queue Service (Amazon SQS)
- The resource APIs are currently provided as a preview. Please be cautious about writing and distributing production-quality code that relies on these resource APIs, especially as the resource APIs may undergo frequent changes during the preview stage.

The following information describes how to download and reference the resource APIs. Links to code examples and related programming concepts for supported AWS services are also provided.

4.1.1 Download and Reference the AWS Resource APIs for .NET

1. If you have an existing project in Visual Studio that you want to use the resource APIs with, and that project is already referencing the AWS .NET library file (`AWSSDK.dll`), you must remove this reference. This reference is set by default if you have the AWS Toolkit for Visual Studio installed and you have created a project based upon one of the AWS project templates (for example, the Visual C# AWS Console Project template). Or, you may have previously set a reference to the library explicitly, which the the SDK typically installs to `drive:\Program Files (x86)\AWS SDK for .NET\bin`. To remove the reference for example in *Solution Explorer* in Visual Studio, in the *References* folder, right-click *AWSSDK* and then click *Remove*.
2. Download the AWS Resource APIs for .NET library file from the resourceAPI-preview branch of the `aws-sdk-net` GitHub repository onto your development machine. To do this, in the binaries folder at that location, download and then unzip the file named dotnet35.zip (for projects that rely on the .NET Framework 3.5) or dotnet45.zip (for projects that rely on the .NET Framework 4.5). Note that because these zip files contains a file that is also named `AWSSDK.dll`, make sure to unzip the file to a location *other* than where your AWS .NET library file is already installed. For example, unzip the file to any location *other* than `drive:\Program Files (x86)\AWS SDK for .NET\bin`. The unzipped contents contain both .NET Framework 3.5 and 4.5 versions of the AWS Resource APIs for .NET library file (`AWSSDK.dll`), which you can set a reference to from your projects.

   Note that after unzipping, there will be three files: `AWSSDK.dll`, `AWSSDK.pdb`, and `AWSSDK.xml`. To enable robust debugging and help within Visual Studio, make sure that these three files remain together in the same folder.
3. From the project in Visual Studio that you want to use the resource APIs with, set a reference to the AWS Resource APIs for .NET library file that you just unzipped. To do this for example in *Solution Explorer* in Visual Studio, right-click the *References* folder; click *Add Reference*; click *Browse*; browse to and select the `AWSSDK.dll` file that you just unzipped; click *Add* and then click *OK*.
4. Import the specific resource APIs in the AWS Resource APIs for .NET that you want to use in your project's code. These APIs typically take the format `Amazon.ServiceName.Resources`, where {ServiceName} is typically some recognizable phrase that corresponds to the specific service.

For example for the AWS Identity and Access Management resource APIs, in C# you would include the following `using` directive at the top of a class file:

```
using Amazon.IdentityManagement.Resources;
```

5. As needed, import any corresponding low-level APIs that the specific resource APIs rely upon. These APIs typically take the format `Amazon.ServiceName.Model` and sometimes also `Amazon.ServiceName`, where {ServiceName} is typically some recognizable phrase that corresponds to the specific service. For example for the AWS Identity and Access Management low-level APIs, in C# you would include the following `using` directives at the top of a class file:

```
using Amazon.IdentityManagement.Model;
// Possibly also the following, depending on which of the resource APIs␣
↪that you use:
using Amazon.IdentityManagement
```

6. Because the resource APIs are currently provided as a preview, you should be cautious about writing production-quality code that relies on them, especially as the resource APIs may undergo frequent changes during the preview stage. However, if you choose to distribute the project anyway, make sure to include a copy of the AWS Resource APIs for .NET library file. To do this for example in *Solution Explorer* in Visual Studio, within the *References* folder, click *AWSSDK*; in the *Properties* window, next to *Copy Local*, select *True* if it is not already selected.

Note: If you distribute a project that has a copy of the resource APIs library file included, and then the resource library APIs change, the only way for your project to include the new changes is to redistribute your project with an updated resource APIs library file copied locally.

4.1.2 Code Examples for Resource APIs

The following links provide code examples for AWS services that support resource-level APIs in the the SDK.

- *CloudFormation*
- *Amazon Glacier*
- *AWS Identity and Access Management (IAM)*
- *Amazon Simple Notification Service (Amazon SNS)*
- *Amazon Simple Queue Service (Amazon SQS)*

4.2 AWS CloudFormation Programming with the AWS SDK for .NET

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

The following information introduces you to the CloudFormation programming models in the the SDK.

4.2.1 Programming Models

The the SDK provides two programming models for working with CloudFormation. These programming models are known as the *low-level* and *resource* models. The following information describes these models, how to use them, and why you would want to use them.

Low-Level APIs

The the SDK provides low-level APIs for programming with CloudFormation. These low-level 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.

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 sdk-net-api-v2.

Resource APIs

The the SDK provides the AWS Resource APIs for .NET for programming with CloudFormation. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with CloudFormation resources as compared to their low-level API counterparts. (For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.)

The following example shows how to use the AWS Resource APIs for .NET to list accessible resources in CloudFormation:

```
// using Amazon.CloudFormation.Resources;

var cf = new CloudFormation();

foreach (var stack in cf.GetStacks())
{
  Console.WriteLine("Stack: {0}", stack.Name);
  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.Resources.

4.3 Amazon DynamoDB Programming with the AWS SDK for .NET

The AWS SDK for .NET supports Amazon DynamoDB, which is a fast NoSQL database service offered by AWS.

The following information introduces you to the DynamoDB programming models and their APIs. There are also links to additional DynamoDB programming resources within the the AWS SDK for .NET.

4.3.1 Amazon DynamoDB Programming with Expressions by Using the AWS SDK for .NET

The following code examples demonstrate how to use the the SDK to program DynamoDB with expressions. *Expressions* denote the attributes that 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 to indicate 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.

Topics

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 that is 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 properties such as `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`).

Additional Resources

For additonal 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

4.3.2 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.

Topics

- *Get Data from a DynamoDB Table in JSON Format*
- *Insert JSON Format Data into a DynamoDB Table*

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 `Document` class's `ToJson` method converts an item from the table into a JSON-formatted string. The item is retrieved through the `Table` class's `GetItem` method. 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 `Table` class's `LoadTable` method 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 `Document` class's `FromJson` method converts a JSON-formatted string into an item. The item is inserted into the table through the `Table` class's `PutItem` method, which uses the instance of the `Document` class that contains the item. To determine the table to insert the item into, the `Table` class's `LoadTable` method 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 `Document` class's `ToJson` method, 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 convert 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 `Document` class's `DecodeBase64Attributes` method 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");
```

Additional Resources

For additional information and examples of programming JSON with DynamoDB with the the SDK, see:

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

4.3.3 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.

Amazon 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:

```
<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 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 that is 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 roles' credentials. For more information, see *Tutorial: Grant Access Using an IAM Role and the AWS SDK for .NET* 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. You can use either the AWS 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 |region_api_default|:

```
{ "Version" : "2012-10-17",
  "Statement" : [
    {
      "Sid" : "1",
      "Effect" : "Allow",
      "Action" : [
        "dynamodb:DeleteItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem" ],
      "Resource" : "arn:aws:dynamodb:us-west-2{<YOUR-AWS-ACCOUNT-ID>}:table/
↪ASP.NET_SessionState"
    }
  ]
}
```

4.3.4 Programming Models

The the SDK provides three different programming models for communicating with DynamoDB. These programming models include the *low-level* model, the *document* model, and the *object persistence* model. The following information describes these models, how to use them, and when you might want to use them.

Topics

- *Low-Level*
- *Document*
- *Object Persistence*

Low-Level

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 example shows how to use the low-level model to create a table in DynamoDB:

```
// 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);
```

In the preceding example, the table is created through the `AmazonDynamoDBClient` class's `CreateTable` method. The `CreateTable` method uses an instance of the `CreateTableRequest` class containing characteristics such as required item attribute names, primary key definition, and throughput capacity. The `CreateTable` method returns an instance of the `CreateTableResponse` class.

Before you begin modifying a table, you should make sure that the table is ready. The following example shows how to use the low-level model to verify that a table in DynamoDB is ready:

```
// 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);
```

In the preceding example, the target table to check is referenced through the `AmazonDynamoDBClient` class's `DescribeTable` method. Every 5 seconds, the code checks the value of the table's `TableStatus` property. When the status is set to `ACTIVE`, then the table is ready to be modified.

The following example shows how to use the low-level model to insert two items into a table in DynamoDB:

```
// 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);
```

In the preceding example, each item is inserted through the `AmazonDynamoDBClient` class's `PutItem` method, using an instance of the `PutItemRequest` class. Each of the two instances of the `PutItemRequest` class takes the name of the table to be inserted into, along with a series of item attribute values.

For more information and examples, see:

- 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

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.

Of the three models, the document model is easier to code against DynamoDB data compared to the low-level programming model. 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 tables, you must use the low-level model. Finally, this model requires you to write more code to store, load, and query .NET objects compared to the object persistence model.

The following example shows how to use the document model to insert an item into a 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);
```

In the preceding example, the item is inserted into the table through the `Table` class's `PutItem` method. 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, the `Table` class's `LoadTable` method is called, specifying an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

The following example shows how to use the document model to get an item from a 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"]);
```

In the preceding example, the item is retrieved through the `Table` class's `GetItem` method. 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 `Table` class's `LoadTable` method uses an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

The preceding example implicitly converts the attribute values for `Id`, `Type`, `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"];
```

For more information about data type conversions with DynamoDB, see DynamoDB Data Types and DynamoDBEntry.

For more information and examples about the DynamoDB document model, see .NET: Document Model.

Object Persistence

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 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 you can use this model to map your .NET classes to the tables.

The following example shows how to define a .NET class that represents an item in a table in DynamoDB:

```
// 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; }
}
```

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

The following example shows how to use an instance of this .NET class to insert an item into a table in DynamoDB:

```
// 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);
```

In the preceding example, the item is inserted through the `DynamoDBContext` class's `Save` method, 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.)

The following example shows how to use an instance of this .NET object to get an item from a table in DynamoDB:

```
// 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);
```

In the preceding example, the item is retrieved through the `DynamoDBContext` class's `Load` method, 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 before, the instance of the `DynamoDBContext` class is initialized with an instance of the `AmazonDynamoDBClient` class.)

For more information and examples, see .NET: Object Persistence Model.

4.3.5 Additional Resources

For additional information and examples of programming DynamoDB with the the SDK, see:

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

4.4 Amazon Elastic Compute Cloud Programming with the AWS SDK for .NET

The AWS SDK for .NET supports Amazon Elastic Compute Cloud (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.

4.4.1 Tutorial: Creating Amazon EC2 Instances with the AWS SDK for .NET

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

The sample code in this tutorial 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, select *New Project* from the *File* menu, and then select *AWS Empty Project*.

Prerequisites

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

Tasks

The following tasks demonstrate how to manage EC2 instances using the AWS SDK for .NET.

- *Create an Amazon EC2 Client Using the the SDK*
- *Create a Security Group Using the the SDK*
- *Create a Key Pair Using the the SDK*
- *Launch an EC2 Instance Using the the SDK*
- *Terminate an EC2 Instance Using the the SDK*

Create an Amazon EC2 Client Using the the SDK

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 that is attached to the profile that 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 in the Amazon Web Services General Reference.

Create a Security Group Using the the SDK

Create a *security group*, which 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 using the Amazon EC2 console or the the SDK.

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.

Contents

Enumerating Your Security Groups

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

To enumerate your security groups for EC2-Classic

Get the complete list of your security groups using DescribeSecurityGroups with no parameters. The following example checks each security group to see whether its name is `my-sample-sg`.

```
string secGroupName = "my-sample-sg";
SecurityGroup mySG = null;

var dsgRequest = new DescribeSecurityGroupsRequest();
var dsgResponse = ec2Client.DescribeSecurityGroups(dsgRequest);
List<SecurityGroup> mySGs = dsgResponse.SecurityGroups;
foreach (SecurityGroup item in mySGs)
{
    Console.WriteLine("Existing security group: " + item.GroupId);
    if (item.GroupName == secGroupName)
    {
        mySG = item;
    }
}
```

To enumerate your security groups for a VPC

To enumerate the security groups for a particular VPC, use DescribeSecurityGroups with a filter. The following example checks each security group for a security group with the name `my-sample-sg-vpc`.

```
string secGroupName = "my-sample-sg-vpc";
SecurityGroup mySG = null;
string vpcID = "vpc-f1663d98";

Filter vpcFilter = new Filter
{
    Name = "vpc-id",
    Values = new List<string>() {vpcID}
};
var dsgRequest = new DescribeSecurityGroupsRequest();
dsgRequest.Filters.Add(vpcFilter);
var dsgResponse = ec2Client.DescribeSecurityGroups(dsgRequest);
List<SecurityGroup> mySGs = dsgResponse.SecurityGroups;
foreach (SecurityGroup item in mySGs)
{
    Console.WriteLine("Existing security group: " + item.GroupId);
    if (item.GroupName == secGroupName)
    {
        mySG = item;
    }
}
```

Creating a Security Group

The examples in this section follow from the examples in the previous section. If the security group doesn't already exist, create it. Note that if you were to specify the same name as an existing security group, `CreateSecurityGroup` throws an exception.

To create a security group for EC2-Classic

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 ID of the new security group from the response and then use DescribeSecurityGroups with the security group ID to get the `SecurityGroup` object for the security group.

```
if (mySG == null)
{
    var newSGRequest = new CreateSecurityGroupRequest()
    {
        GroupName = secGroupName,
        Description = "My sample security group for EC2-Classic"
    };
```

```
    var csgResponse = ec2Client.CreateSecurityGroup(newSGRequest);
    Console.WriteLine();
    Console.WriteLine("New security group: " + csgResponse.GroupId);

    List<string> Groups = new List<string>() { csgResponse.GroupId };
    var newSgRequest = new DescribeSecurityGroupsRequest() { GroupIds =
↪Groups };
    var newSgResponse = ec2Client.DescribeSecurityGroups(newSgRequest);
    mySG = newSgResponse.SecurityGroups[0];
}
```

To create a security group for EC2-VPC

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 ID of the new security group from the response and then use DescribeSecurityGroups with the security group ID to get the `SecurityGroup` object for the security group.

```
if (mySG == null)
{
    var newSGRequest = new CreateSecurityGroupRequest()
    {
        GroupName = secGroupName,
        Description = "My sample security group for EC2-VPC",
        VpcId = vpcID
    };
    var csgResponse = ec2Client.CreateSecurityGroup(newSGRequest);
    Console.WriteLine();
    Console.WriteLine("New security group: " + csgResponse.GroupId);

    List<string> Groups = new List<string>() { csgResponse.GroupId };
    var newSgRequest = new DescribeSecurityGroupsRequest() { GroupIds =
↪Groups };
    var newSgResponse = ec2Client.DescribeSecurityGroups(newSgRequest);
    mySG = newSgResponse.SecurityGroups[0];
}
```

Adding 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 get the public IP address of your local computer using a service. 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 that `mySG` is an existing security group.

To add a rule to a security group

1. Create and initialize an IpPermission object.

```
string ipRange = "0.0.0.0/0";
List<string> ranges = new List<string>() {ipRange};

var ipPermission = new IpPermission()
{
    IpProtocol = "tcp",
    FromPort = 3389,
    ToPort = 3389,
    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 `0.0.0.0/0`, which authorizes network traffic from all IP addresses. This is acceptable for a short time in a test environment, but it's unsafe in a production environment.

2. Create and initialize an AuthorizeSecurityGroupIngressRequest object.

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

   GroupId The ID 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 request object to the AuthorizeSecurityGroupIngress method, which returns an AuthorizeSecurityGroupIngressResponse object.

```
var ingressResponse =  ec2Client.
 ↪AuthorizeSecurityGroupIngress(ingressRequest);
Console.WriteLine("New RDP rule for: " + ipRange);
```

Create a Key Pair Using the the SDK

You must specify a key pair when you launch an EC2 instance and specify the private key of the key pair when you connect to the instance. You can create a key pair or use an existing key pair that you've used when launching other instances. For more information, see Amazon EC2 Key Pairs in the Amazon EC2 User Guide for Windows Instances.

Enumerating Your Key Pairs

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

To enumerate your key pairs

Get the complete list of your key pairs using DescribeKeyPairs with no parameters. The following example checks each key pair to see whether its name is `my-sample-key`.

```
string keyPairName = "my-sample-key";
KeyPairInfo myKeyPair = null;

var dkpRequest = new DescribeKeyPairsRequest();
var dkpResponse = ec2Client.DescribeKeyPairs(dkpRequest);
List<KeyPairInfo> myKeyPairs = dkpResponse.KeyPairs;

foreach (KeyPairInfo item in myKeyPairs)
{
    Console.WriteLine("Existing key pair: " + item.KeyName);
    if (item.KeyName == keyPairName)
    {
        myKeyPair = item;
    }
}
```

Creating a Key Pair and Saving the Private Key

The example in this section follows from the example in the previous section. If the key pair doesn't already exist, create it. Be sure to save the private key now, because you can't retrieve it later.

To create a key pair and save the private key

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 CreateKeyPairResponse object.

The response object includes a CreateKeyPairResult property 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 current directory, using the name of the key pair as the base file name of the `.pem` file.

```
if (myKeyPair == null)
{
    var newKeyRequest = new CreateKeyPairRequest()
    {
        KeyName = keyPairName
    };
    var ckpResponse = ec2Client.CreateKeyPair(newKeyRequest);
    Console.WriteLine();
    Console.WriteLine("New key: " + keyPairName);

    // Save the private key in a .pem file
    using (FileStream s = new FileStream(keyPairName + ".pem", FileMode.
↪Create))
    using (StreamWriter writer = new StreamWriter(s))
    {
        writer.WriteLine(ckpResponse.KeyPair.KeyMaterial);
    }
}
```

Launch an EC2 Instance Using the the SDK

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

Contents

Launching an EC2 Instance

You 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.

To launch an EC2 instance in EC2-Classic

1. Create and initialize a RunInstancesRequest object. Make sure that the AMI, key pair, and security group that you specify exist in the region that 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 = "t1.micro",
    MinCount = 1,
    MaxCount = 1,
    KeyName = keyPairName,
    SecurityGroupIds = groups
};
```

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 *Create a Key Pair Using the the SDK*.

SecurityGroupIds The identifiers of one or more security groups. For more information, see *Create a Security Group Using the the SDK*.

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

   Note that 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";
instanceProfile.Arn = "arn:aws:iam::|ExampleAWSAccountNo2H|:instance-
 ↪profile/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 instances, as 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 that you successfully launched. You can retrieve the ID for each instance from the `Instance` object's `InstanceId` property.

```
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);
}
```

To launch an EC2 instance in a VPC

1. Create and initialize a network interface.

```
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
 ↪<InstanceNetworkInterfaceSpecification>() {eni};
```

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

`SubnetId` The ID of the subnet to launch the instance into.

`GroupIds` One or more security groups. For more information, see *Create a Security Group Using the the SDK*.

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

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

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

var launchRequest = new RunInstancesRequest()
{
    ImageId = amiID,
    InstanceType = "t1.micro",
    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 *Create a Key Pair Using the the SDK*.

`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.

   Note that 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";
instanceProfile.Arn = "arn:aws:iam::|ExampleAWSAccountNo2H|:instance-
↪profile/winapp-instance-role-1";
```

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

```
InstanceProfile = instanceProfile
```

4. Launch the instances by passing the request object to the RunInstances method. Save the IDs of the instances, as 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 that you successfully launched. You can retrieve the ID for each instance from the `Instance` object's `InstanceId` property.

```
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);
}
```

Checking 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.

To check the state of your instance

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 EC2 client's 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);
```

Connecting to Your Running Instance

After an instance is running, you can remotely connect to it using an RDP client on your computer. Before connecting to your instance, you must ensure that the instance's RDP port is open to traffic. To connect, you need the instance ID and the private key for instance's key pair. For more information, see Connecting to Your Windows Instance Using RDP in the Amazon EC2 User Guide for Windows Instances.

When you have finished with your EC2 instance, see *Terminate an EC2 Instance Using the the SDK*.

Terminate an EC2 Instance Using the the SDK

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

To terminate an EC2 instance

Create and initialize a TerminateInstancesRequest object. Set the InstanceIds property to a list of one or more instance IDs. In this example, `instanceIds` is the list that you saved when you launched the instances.

Pass the request object to the client object's TerminateInstances method.

```
var deleteRequest = new TerminateInstancesRequest()
{
    InstanceIds = instanceIds
};
var deleteResponse = ec2Client.TerminateInstances(deleteRequest);
foreach (InstanceStateChange item in deleteResponse.TerminatingInstances)
{
    Console.WriteLine();
    Console.WriteLine("Terminated instance: " + item.InstanceId);
    Console.WriteLine("Instance state: " + item.CurrentState.Name);
}
```

To list the terminated instances

You can use the response object as follows to list the terminated instances.

```
List<InstanceStateChange> terminatedInstances = termResponse.
↪TerminateInstancesResult.TerminatingInstance;
foreach(InstanceStateChange item in terminatedInstances)
{
  Console.WriteLine("Terminated Instance: " + item.InstanceId);
}
```

4.4.2 Tutorial: 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, and 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 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 that 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 usage.

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 that 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 that 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*.

Step 1: Setting Up Your Credentials

To begin using this code sample, you need to populate the `App.config` file with your AWS credentials, which identify you to Amazon Web Services. You specify your credentials in the files `appSettings` element. The preferred way to handle credentials is to create a profile in the SDK Store, which encrypts your credentials and stores them separately from any project. You can then specify the profile by name in the `App.config` file, and the credentials are automatically incorporated into the application. For more information, see *Configuring Your AWS SDK for .NET Application*.

Now that you have configured your settings, you can get started using the code in the example.

Step 2: Setting Up a Security Group

A *security group* acts as a firewall that controls the traffic allowed in and out of a group of instances. By default, an instance is started without any security group, which means that all incoming IP traffic, on any TCP port will be denied. So, before submitting your Spot Request, you will set up a security group that allows the necessary network traffic. For the purposes of this tutorial, we will create a new security group called "GettingStarted" that allows connection using the Windows Remote Desktop Protocol (RDP) from the IP address of the local computer, that is, the computer where you are running the application.

To set up a new security group, you need to include or run the following code sample that sets up the security group programmatically. You only need to run this code once to create the new security group. However, the code is designed so that it is safe to run even if the security group already exists. In this case, the code catches and ignores the "InvalidGroup.Duplicate" exception.

In the code below, we first use *AWSClientFactoryClass* to create an *AmazonEC2* client object. We then create a `CreateSecurityGroupRequest` object with the name, "GettingStarted" and a description for the security group. Finally, we call the `ec2.createSecurityGroup` API to create the group.

```
AmazonEC2 ec2 = AWSClientFactory.CreateAmazonEC2Client();

try
{
    CreateSecurityGroupRequest securityGroupRequest = new␣
↪CreateSecurityGroupRequest();
    securityGroupRequest.GroupName = "GettingStartedGroup";
    securityGroupRequest.GroupDescription = "Getting Started Security Group";

    ec2.CreateSecurityGroup(securityGroupRequest);
}
catch (AmazonEC2Exception ae)
{
```

```
    if (string.Equals(ae.ErrorCode, "InvalidGroup.Duplicate",␣
↪StringComparison.InvariantCulture))
    {
        Console.WriteLine(ae.Message);
    }
    else
    {
        throw;
    }
}
```

To enable access to the group, we create an `ipPermission` object with the IP address set to the CIDR representation of the IP address of the local computer. The "/32" suffix on the IP address indicates that the security group should accept traffic *only* from the local computer. We also configure the `ipPermission` object with the TCP protocol and port 3389 (RDP). You will need to fill in the IP address of the local computer. If your connection to the Internet is mediated by a firewall or some other type of proxy, you will need to determine the external IP address that the proxy uses. One technique is to query a search engine such as Google or Bing with the string: "what is my IP address".

```
// TODO - Change the code below to use your external IP address.
String ipSource = "XXX.XXX.XXX.XX/32";

List<String> ipRanges = new List<String>();
ipRanges.Add(ipSource);

List<IpPermissionSpecification> ipPermissions = new List
↪<IpPermissionSpecification>();
IpPermissionSpecification ipPermission = new IpPermissionSpecification();
ipPermission.IpProtocol = "tcp";
ipPermission.FromPort = 3389;
ipPermission.ToPort = 3389;
ipPermission.IpRanges = ipRanges;
ipPermissions.Add(ipPermission);
```

The final step is to call `ec2.authorizeSecurityGroupIngress` with the name of our security group and the `ipPermission` object.

```
try {
    // Authorize the ports to be used.
    AuthorizeSecurityGroupIngressRequest ingressRequest = new␣
↪AuthorizeSecurityGroupIngressRequest();
    ingressRequest.IpPermissions = ipPermissions;
    ingressRequest.GroupName = "GettingStartedGroup";
    ec2.AuthorizeSecurityGroupIngress(ingressRequest);
} catch (AmazonEC2Exception ae) {
    if (String.Equals(ae.ErrorCode, "InvalidPermission.Duplicate",␣
↪StringComparison.InvariantCulture))
    {
        Console.WriteLine(ae.Message);
    }
    else
    {
```

```
            throw;
        }
    }
```

You can also create the security group using the AWS Toolkit for Visual Studio. Go to the Toolkit for Visual Studio User Guide for more information.

Step 3: 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 the security group we configured previously, so that you can log into the instance if you want to.

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 when it completes. You want to see if you can complete it for less cost than with 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 Rrequest 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 provide a high likelihood that your request will be fulfilled quickly and continue computing until it completes.

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 that 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 Instance Types .

To request a Spot Instance, you simply need to build your request with the parameters we have specified so far. We start by creating a `RequestSpotInstanceRequest` object. The request object requires the number of instances you want to start (2) and the bid price ($0.03). Additionally, you need to set the `LaunchSpecification` for the request, which includes the instance type, AMI ID, and security group you want to use. Once the request is populated, you call the `requestSpotInstances` method on the `AmazonEC2Client` object. An example of how to request a Spot Instance is shown below.

```
RequestSpotInstancesRequest requestRequest = new␣
↪RequestSpotInstancesRequest();

requestRequest.SpotPrice = "0.03";
requestRequest.InstanceCount = 2;

LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.ImageId = "ami-fbf93092";    // latest Windows AMI as of␣
↪this writing
launchSpecification.InstanceType = "t1.micro";

launchSpecification.SecurityGroup.Add("GettingStartedGroup");

requestRequest.LaunchSpecification = launchSpecification;

RequestSpotInstancesResponse requestResult = ec2.
↪RequestSpotInstances(requestRequest);
```

There are other options you can use to configure your Spot Requests. To learn more, see RequestSpotInstances in the AWS SDK for .NET.

Running this code will launch a new Spot Instance Request.

Note: You will be charged for any Spot Instances that are actually launched, so make sure that you cancel any requests and terminate any instances you launch to reduce any associated fees.

Step 4: Determining the State of Your Spot Request

Next, we want to create code to wait until the Spot Request reaches the "active" state before proceeding to the last step. To determine the state of our Spot Request, we poll the describeSpotInstanceRequests method for the state of the Spot Request ID we want to monitor.

The request ID created in Step 2 is embedded in the result of our `requestSpotInstances` request. The following example code gathers request IDs from the `requestSpotInstances` result and uses them to populate the `SpotInstanceRequestId` member of a `describeRequest` object. We will use this object in the next part of the sample.

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResponse requestResult = ec2.
 ↪RequestSpotInstances(requestRequest);

// Create the describeRequest object with all of the request ids
// to monitor (e.g. that we started).
DescribeSpotInstanceRequestsRequest describeRequest = new␣
 ↪DescribeSpotInstanceRequestsRequest();
foreach (SpotInstanceRequest spotInstanceRequest in requestResult.
 ↪RequestSpotInstancesResult.SpotInstanceRequest)
{
    describeRequest.SpotInstanceRequestId.Add(spotInstanceRequest.
 ↪SpotInstanceRequestId);
}
```

```
// Create a variable that will track whether there are any
// requests still in the open state.
bool anyOpen;

// Create a list to store any instances that were activated.
List<String> instanceIds = new List<String>();

do
{
    // Initialize the anyOpen variable to false, which assumes there
    // are no requests open unless we find one that is still open.
    anyOpen = false;
    instanceIds.Clear();

    try
    {
        // Retrieve all of the requests we want to monitor.
        DescribeSpotInstanceRequestsResponse describeResponse = ec2.
 ↪DescribeSpotInstanceRequests(describeRequest);

        // Look through each request and determine if they are all in
        // the active state.
        foreach (SpotInstanceRequest spotInstanceRequest in describeResponse.
 ↪DescribeSpotInstanceRequestsResult.SpotInstanceRequest)
        {
            // If the state is open, it hasn't changed since we attempted
            // to request it. There is the potential for it to transition
            // almost immediately to closed or canceled, so we compare
            // against open instead of active.
            if (spotInstanceRequest.State.Equals("open", StringComparison.
 ↪InvariantCulture))
            {
                anyOpen = true;
                break;
            }
            else if (spotInstanceRequest.State.Equals("active",␣
 ↪StringComparison.InvariantCulture))
            {
```

```
                // Add the instance id to the list we will
                // eventually terminate.
                instanceIds.Add(spotInstanceRequest.InstanceId);
            }
        }
    }
    catch (AmazonEC2Exception e)
    {
        // If we have an exception, ensure we don't break out of
        // the loop. This prevents the scenario where there was
        // blip on the wire.
        anyOpen = true;

        Console.WriteLine(e.Message);
    }

    if (anyOpen)
    {
        // Wait for the requests to go active.
        Console.WriteLine("Requests still in open state, will retry in 60␣
↪seconds.");
        Thread.Sleep((int)TimeSpan.FromMinutes(1).TotalMilliseconds);
    }
} while (anyOpen);
```

If you just ran the code up to this point, your Spot Instance Request would complete—or possibly fail with an error. For the purposes of this tutorial, we'll add some code that cleans up the requests after all of them have transitioned out of the open state.

Step 5: Cleaning up Your Spot Requests and Instances

The final step is to clean up our 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 pay 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.

The following code demonstrates how to cancel your requests.

```
try
{
    // Cancel requests.
    CancelSpotInstanceRequestsRequest cancelRequest = new␣
↪CancelSpotInstanceRequestsRequest();

    foreach (SpotInstanceRequest spotInstanceRequest in requestResult.
↪RequestSpotInstancesResult.SpotInstanceRequest)
    {
        cancelRequest.SpotInstanceRequestId.Add(spotInstanceRequest.
↪SpotInstanceRequestId);
```

```
    }

    ec2.CancelSpotInstanceRequests(cancelRequest);
}
catch (AmazonEC2Exception e)
{
    // Write out any exceptions that may have occurred.
    Console.WriteLine("Error cancelling instances");
    Console.WriteLine("Caught Exception: " + e.Message);
    Console.WriteLine("Reponse Status Code: " + e.StatusCode);
    Console.WriteLine("Error Code: " + e.ErrorCode);
    Console.WriteLine("Request ID: " + e.RequestId);
}
}
```

To terminate any outstanding instances, we use the instanceIds array, which we populated with the instance IDs of those instances that transitioned to the active state. We terminate these instances by assigning this array to the `InstanceId` member of a `TerminateInstancesRequest` object, then passing that object to the `ec2.TerminateInstances` API.

```
if (instanceIds.Count > 0)
{
    try
    {
        TerminateInstancesRequest terminateRequest = new␣
↪TerminateInstancesRequest();
        terminateRequest.InstanceId = instanceIds;

        ec2.TerminateInstances(terminateRequest);
    }
    catch (AmazonEC2Exception e)
    {
        Console.WriteLine("Error terminating instances");
        Console.WriteLine("Caught Exception: " + e.Message);
        Console.WriteLine("Reponse Status Code: " + e.StatusCode);
        Console.WriteLine("Error Code: " + e.ErrorCode);
        Console.WriteLine("Request ID: " + e.RequestId);
    }
}
```

Conclusion

Congratulations! You have just completed the getting started tutorial for developing Spot Instance software with the AWS SDK for .NET.

4.5 Amazon Glacier Programming with the AWS SDK for .NET

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.

The following information introduces you to the Amazon Glacier programming models in the the SDK.

4.5.1 Programming Models

The the SDK provides three programming models for working with Amazon Glacier. These programming models are known as the *low-level*, *high-level*, and *resource* models. The following information describes these models, why you would want to use them, and how to use them.

Low-Level APIs

The the SDK provides low-level APIs for programming with Amazon Glacier. These low-level APIs map closely 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 additional 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.Model` in the AWS SDK for .NET API Reference.

High-Level APIs

The the SDK 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:

- 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.

Resource APIs

The the SDK provides the AWS Resource APIs for .NET for programming with Amazon Glacier. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with Amazon Glacier resources as compared to their low-level and high-level API counterparts. (For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.)

The following example shows how to use the AWS Resource APIs for .NET to list accessible vaults in Amazon Glacier:

```
// using Amazon.Glacier.Resources;
// using Amazon.Runtime.Resources;

var g = new Glacier();

foreach (var vault in g.GetVaults())
{
  Console.WriteLine("Vault: {0}", vault.Name);
  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 n = vault.GetNotification();

    Console.WriteLine("  Notifications:");
    Console.WriteLine("    Topic: {0}", n.SNSTopic);

    var events = n.Events;

    if (events.Any())
    {
      Console.WriteLine("    Events:");

      foreach (var e in events)
      {
        Console.WriteLine("{0}", e);
      }
    }
    else
    {
      Console.WriteLine("    No events set.");
    }

  }
```

```
  catch (ResourceLoadException)
  {
    Console.WriteLine("    No notifications set.");
  }

  var jobs = vault.GetJobs();

  if (jobs.Any())
  {
    Console.WriteLine("  Jobs:");

    foreach (var job in jobs)
    {
      Console.WriteLine("    For job ID: {0}",
        job.Id);
      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 related API reference information, see Amazon.Glacier.Resources.

4.6 AWS Identity and Access Management Programming with the AWS SDK for .NET

The AWS SDK for .NET supports AWS Identity and Access Management (IAM), which is a web service that enables Amazon Web Services (AWS) customers to manage users and user permissions in AWS.

The following information introduces you to the IAM programming models in the the SDK. There are also links to additional IAM programming resources within the the SDK.

4.6.1 AWS Identity and Access Management Code Examples with the AWS Resource APIs for .NET

The following code examples demonstrate how to program with IAM by using the AWS Resource APIs for .NET.

The AWS Resource APIs for .NET provide a resource-level programming model that enables you to write code to work more directly with resources that are managed by AWS services. For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.

The AWS Resource APIs for .NET are currently provided as a preview. This means that these resource APIs may frequently change in response to customer feedback, and these changes may happen without advance notice. Until these resource APIs exit the preview stage, please be cautious about writing and distributing production-quality code that relies on them.

Topics

Get User Account Information

The following example displays information about an existing user account, including its associated groups, policies, and access key IDs:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;
```

```
var iam = new IdentityManagementService();

try
{
  var user = iam.GetUserByName("DemoUser");

  Console.WriteLine("For user {0}:", user.Name);
  Console.WriteLine("  In groups:");

  foreach (var group in user.GetGroups())
  {
    Console.WriteLine("    {0}", group.Name);
  }

  Console.WriteLine("  Policies:");

  foreach (var policy in user.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  Access keys:");

  foreach (var accessKey in user.GetAccessKeys())
  {
    Console.WriteLine("    {0}", accessKey.Id);
  }
}
catch (NoSuchEntityException)
{
  Console.WriteLine("User 'DemoUser' does not exist.");
}
```

The following example displays a list of all accessible user accounts. For each user account, its associated groups, policies, and access key IDs are also displayed:

```
// using Amazon.IdentityManagement.Resources;

var iam = new IdentityManagementService();
var users = iam.GetUsers();

foreach (var user in users)
{
  Console.WriteLine("For user {0}:", user.Name);
  Console.WriteLine("  In groups:");

  foreach (var group in user.GetGroups())
  {
    Console.WriteLine("    {0}", group.Name);
  }

  Console.WriteLine("  Policies:");
```

```
  foreach (var policy in user.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  Access keys:");

  foreach (var accessKey in user.GetAccessKeys())
  {
    Console.WriteLine("    {0}", accessKey.Id);
  }
}
```

Get Group Information

The following example displays information about an existing group, including its associated policies and user accounts:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var group = iam.GetGroupByName("DemoGroup");

  Console.WriteLine("For group {0}:", group.Name);
  Console.WriteLine("  Policies:");

  foreach (var policy in group.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  Users:");

  foreach (var user in group.GetUsers())
  {
    Console.WriteLine("    {0}", user.Name);
  }
}
catch (NoSuchEntityException)
{
  Console.WriteLine("Group 'DemoGroup' does not exist.");
}
```

The following example displays a list of all accessible groups. For each group, its associated policies and user accounts are also displayed:

```
// using Amazon.IdentityManagement.Resources;

var iam = new IdentityManagementService();
var groups = iam.GetGroups();

foreach (var group in groups)
{
  Console.WriteLine("For group {0}:", group.Name);
  Console.WriteLine("  Policies:");

  foreach (var policy in group.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  Users:");

  foreach (var user in group.GetUsers())
  {
    Console.WriteLine("    {0}", user.Name);
  }
}
```

Get Role Information

The following example displays information about an existing role, including its associated policies and instance profiles:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var role = iam.GetRoleByName("DemoEC2");

  Console.WriteLine("For role {0}:", role.Name);
  Console.WriteLine("  Policies:");

  foreach (var policy in role.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  InstanceProfiles:");

  foreach (var instanceProfile in role.GetInstanceProfiles())
  {
    Console.WriteLine("    {0}", instanceProfile.Name);
  }
```

```
}
catch (NoSuchEntityException)
{
  Console.WriteLine("Role 'DemoEC2' does not exist.");
}
```

The following example displays a list of all accessible roles. For each role, its associated policies and instance profiles are also displayed:

```
// using Amazon.IdentityManagement.Resources;

var iam = new IdentityManagementService();
var roles = iam.GetRoles();

foreach (var role in roles)
{
  Console.WriteLine("For role {0}:", role.Name);
  Console.WriteLine("  Policies:");

  foreach (var policy in role.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  InstanceProfiles:");

  foreach (var instanceProfile in role.GetInstanceProfiles())
  {
    Console.WriteLine("    {0}", instanceProfile.Name);
  }
}
```

Create a User Account

The following example creates a new user account and then displays some information about it:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var user = iam.CreateUser("DemoUser");

  Console.WriteLine("User Name = '{0}', ARN = '{1}'",
    user.Name, user.Arn);
}
catch (EntityAlreadyExistsException)
{
  Console.WriteLine("User 'DemoUser' already exists.");
```

```
}
```

Create a Group

The following example creates a new group and then confirms whether the group was successfully created:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var group = iam.CreateGroup("DemoGroup");

  Console.WriteLine(group.Name + " was created.");
}
catch (EntityAlreadyExistsException)
{
  Console.WriteLine("Group 'DemoGroup' already exists.");
}
```

Create a Role

The following example creates a new role and then confirms whether the group was successfully created.

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();
// GenerateAssumeRolePolicy() is a custom method.
string assumeRole = GenerateAssumeRolePolicy();

try
{
  var role = iam.CreateRole(new CreateRoleRequest
  {
    RoleName = "DemoEC2",
    AssumeRolePolicyDocument = assumeRole
  });

  Console.WriteLine(role.Name + " was created.");
}
catch (EntityAlreadyExistsException)
{
  Console.WriteLine("Role 'DemoEC2' already exists.");
}
```

The preceding example relies on the following example to create the new policy.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating a policy document. However, this example is presented for completeness:

```
public static string GenerateAssumeRolePolicy()
{
  // using Amazon.Auth.AccessControlPolicy;

  // Create a policy that looks like this:
  /*
  {
    "Version": "2012-10-17",
    "Id": "DemoEC2Trust",
    "Statement": [
      {
        "Sid": "DemoEC2TrustStatement",
        "Effect": "Allow",
        "Principal": {
          "Service": "ec2.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
      }
    ]
  }
  */

  var action = new ActionIdentifier("sts:AssumeRole");
  var actions = new List<ActionIdentifier>();

  actions.Add(action);

  var principal = new Principal("ec2.amazonaws.com")
  {
    Provider = "Service"
  };
  var principals = new List<Principal>();

  principals.Add(principal);

  var statement = new Statement(Statement.StatementEffect.Allow)
  {
    Actions = actions,
    Id = "DemoEC2TrustStatement",
    Principals = principals
  };
  var statements = new List<Statement>();

  statements.Add(statement);

  var policy = new Policy
  {
    Id = "DemoEC2Trust",
    Version = "2012-10-17",
    Statements = statements
  };
```

```
  return policy.ToJson();
}
```

Add a User Account to a Group

The following example adds an existing user account to an existing group and then displays a list of the group's associated user accounts:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var group = iam.GetGroupByName("DemoGroup");

  group.AddUser("DemoUser");

  Console.WriteLine("Users in group {0}:", group.Name);

  foreach (var user in group.GetUsers())
  {
    Console.WriteLine("  {0}", user.Name);
  }

}
catch (NoSuchEntityException)
{
  Console.WriteLine("Group 'DemoGroup' or " +
    "user 'DemoUser' does not exist.");
}
```

Add a Policy to a User Account, Group, or Role

Add a Policy to a User Account

The following example creates a new policy, adds the new policy to an existing user account, and then displays a list of the user account's associated policies:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var user = iam.GetUserByName("DemoUser");
```

```
  // GenerateUserPolicyDocument() is a custom method.
  string policyDoc = GenerateUserPolicyDocument();

  user.CreatePolicy(policyDoc, "ListDeploymentsPolicy");

  Console.WriteLine("Policies for user {0}:", user.Name);

  foreach (var policyItem in user.GetPolicies())
  {
    Console.WriteLine("  {0}", policyItem.Name);
  }

}
catch (NoSuchEntityException)
{
  Console.WriteLine("User 'DemoUser' does not exist.");
}
```

The preceding example relies on the following example to create the new policy.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating a policy document. However, this example is presented for completeness:

```
public static string GenerateUserPolicyDocument()
{
  // using Amazon.Auth.AccessControlPolicy;

  // Create a policy that looks like this:
  /*
  {
    "Version" : "2012-10-17",
    "Id"  : "ListDeploymentsPolicy",
    "Statement" : [
      {
        "Sid" : "ListDeploymentsStatement",
        "Effect" : "Allow",
        "Action" : "codedeploy:ListDeployments",
        "Resource" : "*"
      }
    ]
  }
  */

  var action = new ActionIdentifier("codedeploy:ListDeployments");
  var actions = new List<ActionIdentifier>();

  actions.Add(action);

  var resource = new Resource("*");
  var resources = new List<Resource>();

  resources.Add(resource);
```

```
  var statement = new Statement(Statement.StatementEffect.Allow)
  {
    Actions = actions,
    Id = "ListDeploymentsStatement",
    Resources = resources
  };
  var statements = new List<Statement>();

  statements.Add(statement);

  var policy = new Policy
  {
    Id = "ListDeploymentsPolicy",
    Version = "2012-10-17",
    Statements = statements
  };

  return policy.ToJson();
}
```

Add a Policy to a Group

The following example creates a new policy, adds the new policy to an existing group, and then displays a list of the group's associated policies:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var group = iam.GetGroupByName("DemoGroup");
  // GenerateGroupPolicyDocument() is a custom method.
  string policyDoc = GenerateGroupPolicyDocument();

  group.CreatePolicy(policyDoc, "ListDeploymentConfigsPolicy");

  Console.WriteLine("Policies for group {0}:", group.Name);

  foreach (var policyItem in group.GetPolicies())
  {
    Console.WriteLine("  {0}", policyItem.Name);
  }

}
catch (NoSuchEntityException)
{
  Console.WriteLine("Group 'DemoGroup' does not exist.");
}
```

The preceding example relies on the following example to create the new policy.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating a policy document. However, this example is presented for completeness:

```
public static string GenerateGroupPolicyDocument()
{
  // using Amazon.Auth.AccessControlPolicy;

  // Create a policy that looks like this:
  /*
  {
    "Version" : "2012-10-17",
    "Id": "ListDeploymentConfigsPolicy",
    "Statement" : [
      {
        "Sid" : "ListDeploymentConfigsStatement",
        "Effect" : "Allow",
        "Action" : "codedeploy:ListDeploymentConfigs",
        "Resource" : "*"
      }
    ]
  }
  */

  var action = new ActionIdentifier("codedeploy:ListDeploymentConfigs");
  var actions = new List<ActionIdentifier>();

  actions.Add(action);

  var resource = new Resource("*");
  var resources = new List<Resource>();

  resources.Add(resource);

  var statement = new Statement(Statement.StatementEffect.Allow)
  {
    Actions = actions,
    Id = "ListDeploymentConfigsStatement",
    Resources = resources
  };
  var statements = new List<Statement>();

  statements.Add(statement);

  var policy = new Policy
  {
    Id = "ListDeploymentConfigsPolicy",
    Version = "2012-10-17",
    Statements = statements
  };

  return policy.ToJson();
}
```

Add a Policy to a Role

The following example creates a new policy and then adds the new policy to an existing role.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support adding a policy to a role. However, this example is presented for completeness:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

var client = new AmazonIdentityManagementServiceClient();
// GenerateRolePolicyDocument() is a custom method.
string policyDoc = GenerateRolePolicyDocument();

var request = new PutRolePolicyRequest
{
  RoleName = "DemoEC2",
  PolicyName = "DemoEC2Permissions",
  PolicyDocument = policyDoc
};

try
{
  client.PutRolePolicy(request);
}
catch (NoSuchEntityException)
{
  Console.WriteLine
    ("Role 'DemoEC2' or policy 'DemoEC2Permissions' does not exist.");
}
```

The preceding example relies on the following example to create the new policy.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating a policy document. However, this example is presented for completeness:

```
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 Statement(Statement.StatementEffect.Allow)
  {
    Actions = actions,
    Id = "DemoEC2PermissionsStatement",
    Resources = resources
  };
  var statements = new List<Statement>();

  statements.Add(statement);

  var policy = new Policy
  {
    Id = "DemoEC2Permissions",
    Version = "2012-10-17",
    Statements = statements
  };

  return policy.ToJson();
}
```

Create an Access Key for a User Account

The following example creates an access key for a user account and then displays the access key's ID and secret access key:

```
// using Amazon.IdentityManagement.Resources;
// using Amazon.IdentityManagement.Model;

var iam = new IdentityManagementService();

try
{
  var user = iam.GetUserByName("DemoUser");
```

```
  var accessKey = user.CreateAccessKey();

  Console.WriteLine("For user {0}:", user.Name);
  Console.WriteLine("  Access key = {0}", accessKey.Id);
  // This is the only time that the secret access key will be displayed.
  Console.WriteLine("  Secret access key = {0}", accessKey.SecretAccessKey);
}
catch (NoSuchEntityException)
{
  Console.WriteLine("User 'DemoUser' does not exist.");
}
catch (LimitExceededException)
{
  Console.WriteLine("You can have only 2 access keys per user.");
}
```

Create a Login Profile for a User Account

The following example creates a login profile for a user account.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating a login profile for a user account. However, this example is presented for completeness:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

var client = new AmazonIdentityManagementServiceClient();
var request = new CreateLoginProfileRequest
{
  UserName = "DemoUser",
  Password = "ksdD9JHm",
  PasswordResetRequired = true
};

try
{
  client.CreateLoginProfile(request);
}
catch (NoSuchEntityException)
{
  Console.WriteLine("User 'DemoUser' doesn't exist.");
}
```

Create an Instance Profile

The following example creates an instance profile.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support creating an instance profile. However, this example is presented for completeness:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

var client = new AmazonIdentityManagementServiceClient();
var request = new CreateInstanceProfileRequest
{
  InstanceProfileName = "DemoEC2-InstanceProfile"
};

try
{
  client.CreateInstanceProfile(request);
}
catch (EntityAlreadyExistsException)
{
  Console.WriteLine(
    "The instance profile 'DemoEC2-InstanceProfile' already exists.");
}
```

Attach an Instance Profile to a Role

The following example attaches an instance profile to a role.

The following example doesn't use the AWS Resource APIs for .NET, as the resource APIs currently don't support attaching an instance profile to a role. However, this example is presented for completeness:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

var client = new AmazonIdentityManagementServiceClient();
var request = new AddRoleToInstanceProfileRequest
{
  RoleName = "DemoEC2",
  InstanceProfileName = "DemoEC2-InstanceProfile"
};

try
{
  client.AddRoleToInstanceProfile(request);
}
catch (NoSuchEntityException)
{
  Console.WriteLine(
    "The role 'DemoEC2' or the instance profile " +
    "'DemoEC2-InstanceProfile' does not exist.");
}
catch (LimitExceededException)
{
  Console.WriteLine("The role 'DemoEC2' already has " +
    "an instance profile attached.");
}
```

4.6.2 Tutorial: Grant Access Using an IAM Role and the AWS SDK for .NET

All requests to AWS must be cryptographically signed using credentials issued by AWS. Therefore, you need a strategy for managing credentials for software that runs on Amazon EC2 instances. You must distribute, store, and rotate these credentials in a way that keeps them secure but also accessible to the software.

We designed IAM roles so that you can effectively manage AWS credentials for software running on EC2 instances. You create an IAM role and configure it with the permissions that the software requires. For more information about the benefits of this approach, 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 does not find credentials in `App.config`, it retrieves temporary credentials that have the same permissions as those associated with the IAM role. The credentials are retrieved from instance metadata. The credentials are stored by the constructor on behalf of the customer 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 walkthrough uses a sample program that retrieves an object from Amazon S3 using the AWS credentials that you've configured. Next, we create an IAM role to provide the AWS credentials. Finally, we launch an instance with an IAM role that provides the AWS credentials to the sample program running on the instance.

Walkthrough

- *Create a Sample that Retrieves an Object from Amazon S3*
- *Create an IAM Role*
- *Launch an EC2 Instance and Specify the IAM Role*
- *Run the Sample Program on the EC2 Instance*

Create a Sample that Retrieves an Object from Amazon S3

The following sample code retrieves an object from Amazon S3. It requires a text file in an Amazon S3 bucket that you have access to. For more information about creating an Amazon S3 bucket and uploading an object, see the Amazon S3 Getting Started Guide. It also requires AWS credentials that provide you with access to the Amazon S3 bucket. For more information, see *Configuring AWS Credentials*.

```
using System;
using System.Collections.Specialized;
using System.IO;

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

namespace s3.amazon.com.docsamples.retrieveobject
{
  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);
        }
      }
    }
}
```

To test the sample code

1. Open Visual Studio and create an AWS Console project.
2. Replace the code in the `Program.cs` file with the sample code.
3. Replace `bucket-name` with the name of your Amazon S3 bucket and `folder/file-name.txt` with the name of a text file in the bucket.
4. 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, ensure that you are using credentials that provide you with access to the bucket.
5. (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.

To create the IAM role

1. Open the IAM console.
2. In the navigation pane, click *Roles*, and then click *Create New Role*.
3. Enter a name for the role, and then click *Next Step*. Remember this name, as you'll need it when you launch your EC2 instance.
4. Under *AWS Service Roles*, select *Amazon EC2*. Under *Select Policy Template*, select *Amazon S3 Read Only Access*. Review the policy and then click *Next Step*.
5. Review the role information and then click *Create Role*.

Launch an EC2 Instance and Specify the IAM Role

You can launch an EC2 instance with an IAM role using the Amazon EC2 console or the the SDK.

- To launch an EC2 instance 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, click *Edit instance details*. In *IAM role*, specify the IAM role that you created previously. Complete the procedure as directed. Notice that you'll need to create or use an existing security group and key pair in order to connect to the instance.

- To launch an EC2 instance with an IAM role using the the SDK, see *Launch an EC2 Instance Using the the SDK*.

Note that 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 (as described in Connecting to an Amazon EC2 Instance in the Toolkit for Visual Studio User Guide) 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.

To run the sample program on the EC2 instance

1. Open the Amazon EC2 console.
2. Get the password for your EC2 instance as follows:

1. In the navigation pane, click *Instances*. Select the instance, and then click *Connect*.
2. In the *Connect To Your Instance* dialog box, click *Get Password*. (It will take a few minutes after the instance is launched before the password is available.)
3. Click *Browse* and navigate to the private key file you created when you launched the instance. Select the file and click *Open* to copy the entire contents of the file into contents box.
4. Click *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 previously with the actual password.

5. 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 as follows:

1. Click *Download Remote Desktop File*. When your browser prompts you to do so, save the `.rdp` file. When you have finished, you can click *Close* to dismiss the *Connect To Your Instance* dialog box.
2. Navigate to your downloads directory, right-click the `.rdp` file, and then select *Edit*. On the *Local Resources* tab, under *Local devices and resources*, click *More*. Select *Drives* to make your local drives available to your instance, and then click *OK*.
3. Click *Connect* to connect to your instance. You may get a warning that the publisher of the remote connection is unknown.
4. Log in to the instance as prompted, using the default *Administrator* account and the default administrator password that you recorded or copied previously.

   Sometimes copying and pasting content can corrupt data. If you encounter a "Password Failed" error when you log 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 both the program and the AWS assembly (`AWSSDK.dll`) from your local drive to the instance.
5. Run the program and verify that it succeeds because it uses the credentials provided by the IAM role.

```
Retrieving (GET) an object
```

4.6.3 Programming Models

The the SDK provides two programming models for working with IAM. These programming models are known as the *low-level* model and the *resource* model. The following information describes these models and how to use them.

Low-Level APIs

The the SDK provides low-level APIs for programming with IAM. These low-level 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.

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:

```
// using Amazon.IdentityManagement;
// using Amazon.IdentityManagement.Model;

var client = new AmazonIdentityManagementServiceClient();
var requestUsers = new ListUsersRequest();
var responseUsers = client.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 = client.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 = client.ListUserPolicies(requestPolicies);

  foreach (var policy in responsePolicies.PolicyNames)
  {
    Console.WriteLine("    {0}", policy);
  }

  var requestAccessKeys = new ListAccessKeysRequest
  {
    UserName = user.UserName
  };
  var responseAccessKeys = client.ListAccessKeys(requestAccessKeys);

  Console.WriteLine("  Access keys:");

  foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
  {
    Console.WriteLine("    {0}", accessKey.AccessKeyId);
  }
}
```

For additional examples, see *Tutorial: Grant Access Using an IAM Role and the AWS SDK for .NET*.

For related API reference information, see Amazon.IdentityManagement and Amazon.IdentityManagement.Model.

Resource APIs

The the SDK provides the AWS Resource APIs for .NET for programming with IAM. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with IAM resources as compared to their low-level API counterparts. (For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.)

The AWS Resource APIs for .NET are currently provided as a preview. This means that these resource APIs may frequently change in response to customer feedback, and these changes may happen without advance notice. Until these resource APIs exit the preview stage, please be cautious about writing and distributing production-quality code that relies on them.

The following example shows how to use the AWS Resource APIs for .NET to list accessible user accounts in IAM. For each user account, its associated groups, policies, and access key IDs are also listed:

```
// using Amazon.IdentityManagement.Resources;

var iam = new IdentityManagementService();
var users = iam.GetUsers();

foreach (var user in users)
{
  Console.WriteLine("For user {0}:", user.Name);
  Console.WriteLine("  In groups:");

  foreach (var group in user.GetGroups())
  {
    Console.WriteLine("    {0}", group.Name);
  }

  Console.WriteLine("  Policies:");

  foreach (var policy in user.GetPolicies())
  {
    Console.WriteLine("    {0}", policy.Name);
  }

  Console.WriteLine("  Access keys:");

  foreach (var accessKey in user.GetAccessKeys())
  {
    Console.WriteLine("    {0}", accessKey.Id);
  }
}
```

For additional examples, see *AWS Identity and Access Management Code Examples with the AWS Resource APIs for .NET*.

For related API reference information, see Amazon.IdentityManagement.

4.7 Amazon Route 53 Programming with the AWS SDK for .NET

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 that you are already familiar with how to use Amazon Route 53 and have already installed the AWS SDK for .NET. For more information on Amazon Route 53, see the Amazon Route 53 Developer Guide. For information on 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 what 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 that it is complete.

The example is a simple console application that shows how to use the the SDK to implement this procedure for a basic record set.

To run this example

1. In the Visual Studio *File* menu, click *New* and then click *Project*.
2. Select 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 that the region is set to US East (Northern 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 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. 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 that 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 example uses the default constructor to create the object, which implicitly specifies the application's default profile and region. Other constructors allow you to override either or both default values.

[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 following request parameters. There are also two optional parameters that aren't used by this example.

   `Name` (Required) The domain name that you want to register, `www.example.com` for this example. This domain name is intended only for examples and can't be registered with a domain name registrar for an actual site, 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 a variety of 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 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 that you want to update. The group of properties that 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 that 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 for each resource record set, and set the following properties.

   `ResourceRecordSet` The `ResourceRecordSet` object that you created in the previous step.

   `Action` The action to be taken for this resource record set: `CREATE`, `DELETE`, or `UPSERT`. For more information on 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 client object's ChangeResourceRecordSets method. It returns a ChangeResourceRecordSetsResponse object, which contains a request ID that 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 has completed 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 client object's GetChange method. `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.

4.8 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 Simple Storage Service.

The following links provide examples of programming Amazon S3 with the the SDK:

- 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

4.9 Amazon Simple Notification Service Programming with the AWS SDK for .NET

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 information introduces you to the Amazon SNS programming models in the the SDK.

4.9.1 Programming Models

The the SDK provides two programming models for working with Amazon SNS. These programming models are known as the *low-level* and *resource* models. The following information describes these models, how to use them, and why you would want to use them.

Low-Level APIs

The the SDK provides low-level APIs for programming with Amazon SNS. These low-level 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.

The following example shows how to use the low-level APIs to list accessible topics in Amazon SNS:

```
// 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..

Resource APIs

The the SDK provides the AWS Resource APIs for .NET for programming with Amazon SNS. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with Amazon SNS resources as compared to their low-level API counterparts. (For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.)

The following example shows how to use the AWS Resource APIs for .NET to list accessible topics in Amazon SNS:

```
// using Amazon.SimpleNotificationService.Resources;

var sns = new SimpleNotificationService();
var topics = sns.GetTopics();

if (topics.Any())
{
  Console.WriteLine("Topics:");

  foreach (var topic in topics)
  {
    Console.WriteLine("  Topic ARN: {0}", topic.Arn);

    if (topic.Attributes.Count > 0)
    {
      Console.WriteLine("    Attributes:");

      foreach (var attr in topic.Attributes)
      {
        Console.WriteLine("{0} = {1}", attr.Key, attr.Value);
      }

    }

  }

}

var subs = sns.GetSubscriptions();

if (subs.Any())
{
  Console.WriteLine("Subscriptions:");

  foreach (var sub in subs)
  {
    Console.WriteLine("  Subscription ARN: {0}", sub.Arn);

    var attrs = sub.Attributes;

    if (attrs.Any())
    {
```

```
      Console.WriteLine("    Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("{0} = {1}", attr.Key, attr.Value);
      }
    }

  }

}
```

For related API reference information, see Amazon.SNS.Resources.

4.10 Amazon Simple Queue Service Programming with the AWS SDK for .NET

The AWS SDK for .NET supports Amazon Simple Queue Service (Amazon SQS), which is a messaging queue service that handles message or workflows between other components in a system. For more information, see the SQS Getting Started Guide.

The following information introduces you to the Amazon SQS programming models in the the SDK.

4.10.1 Programming Models

The the SDK provides two programming models for working with Amazon SQS. These programming models are known as the *low-level* and *resource* models. The following information describes these models, how to use them, and why you would want to use them.

Low-Level APIs

The the SDK provides low-level APIs for programming with Amazon SQS. 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.

The following example shows how to use the APIs to list accessible queues in Amazon SQS:

```
// using Amazon.SQS;
// using Amazon.SQS.Model;

var client = new AmazonSQSClient();

// List all queues that start with "aws".
var request = new ListQueuesRequest
{
  QueueNamePrefix = "aws"
};
```

```
var response = client.ListQueues(request);
var urls = response.QueueUrls;

if (urls.Any())
{
  Console.WriteLine("Queue URLs:");

  foreach (var url in urls)
  {
    Console.WriteLine("  " + url);
  }
}
else
{
  Console.WriteLine("No queues.");
}
```

For additional examples, see the following:

- *Create an Amazon SQS Client*
- *Create an Amazon SQS Queue*
- *Send an Amazon SQS Message*
- *Receive a Message from an Amazon SQS Queue*
- *Delete a Message from an Amazon SQS Queue*

For related API reference information, see `Amazon.SQS`, `Amazon.SQS.Model`, and `Amazon.SQS.Util` in the AWS SDK for .NET Reference.

Resource APIs

The the SDK provides the AWS Resource APIs for .NET for programming with Amazon SQS. These resource APIs provide a resource-level programming model that enables you to write code to work more directly with Amazon SQS resources as compared to their low-level API counterparts. (For more information about the AWS Resource APIs for .NET, including how to download and reference these resource APIs, see *Programming with the AWS Resource APIs for .NET*.)

The following example shows how to use the AWS Resource APIs for .NET to list accessible queues in Amazon SQS

```
// using Amazon.SQS.Resources;

var sqs = new SQS();

// List all queues that start with "aws".
var queues = sqs.GetQueues("aws");

if (queues.Any())
{
  Console.WriteLine("Queue URLs:");
```

```
  foreach (var queue in queues)
  {
    Console.WriteLine("  " + queue.Url);
  }
}
else
{
  Console.WriteLine("No queues.");
}
```

For related API reference information, see Amazon.SQS.Resources.

Creating and Using an Amazon SQS Queue with the AWS SDK for .NET

This topic demonstrates how to use the AWS SDK for .NET to create and use an Amazon SQS queue.

The sample code in this topic is written in C#, but you can use the AWS SDK for .NET with any language that is compatible with the Microsoft .NET Framework.

Create an Amazon SQS Client

You will 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 on credentials and profiles, see *Configuring Your AWS SDK for .NET Application*.

```
<?xml version="1.0"?>
  <configuration>
    <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    </startup>
    <configSections>
      <section name="aws" type="Amazon.AWSSection, AWSSDK"/>
    </configSections> <aws profileName="{my_profile}"/>
  </configuration>
```

After you create this file, you are ready to create and initialize your Amazon SQS client.

To create and initialize an Amazon SQS client

1. Create and initialize an AmazonSQSConfig instance, and set the ServiceURL property with the protocol and service endpoint, as follows:

```
AmazonSQSConfig amazonSQSConfig = new AmazonSQSConfig();

amazonSQSConfig.ServiceURL = "http://sqs.us-west-2.amazonaws.com";
```

```
AmazonSQSConfig amazonSQSConfig = new AmazonSQSConfig();

amazonSQSConfig.ServiceURL = "http://sqs.cn-north-1.amazonaws.com";
```

The AWS SDK for .NET uses US East (N. Virginia) Region as the default region if you do not specify a region in your code. However, the AWS Management Console uses US West (Oregon) Region as its default. Therefore, when using the AWS Management Console in conjunction with your development, be sure to specify the same region in both your code and the console.

The AWS SDK for .NET uses China (Beijing) Region as the default region if you do not specify a region in your code. Therefore, when using the AWS Management Console in conjunction with your development, be sure to specify that region in both your code and the console.

Go to Regions and Endpoints for the current list of regions and corresponding endpoints for each of the services offered by AWS.

2. Use the `AmazonSQSConfig` instance to create and initialize an AmazonSQSClient instance, as follows:

```
amazonSQSClient = new AmazonSQSClient(amazonSQSConfig);
```

You can now use the client to create an Amazon SQS queue. For information about creating a queue, see *Create an Amazon SQS Queue*.

Create an Amazon SQS Queue

You can use the AWS SDK for .NET to programmatically create an Amazon SQS queue. Creating an Amazon SQS Queue is an administrative task. You can create a queue by using the AWS Management Console instead of creating a queue programmatically.

To 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:

```
CreateQueueRequest createQueueRequest =
    new CreateQueueRequest();

createQueueRequest.QueueName = "MySQSQueue";
createQueueRequest.DefaultVisibilityTimeout = 10;
```

Your queue name must only be composed of 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, go to Visibility Timeout. For more information about different queue attributes you can set, go to SetQueueAttributes.

2. After you create the request, pass it as a parameter to the CreateQueue method. The method returns a CreateQueueResponse object, as follows:

```
CreateQueueResponse createQueueResponse =
    amazonSQSClient.CreateQueue(createQueueRequest);
```

For information about how queues work in Amazon SQS, go to How SQS Queues Work.

For information about your queue URL, see *Amazon SQS Queue URLs*.

Amazon SQS Queue URLs

You require the 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}
```

For information on sending a message to a queue, see *Send an Amazon SQS Message*.

For information about receiving messages from a queue, see *Receive a Message from an Amazon SQS Queue*.

For information about deleting messages from a queue, see *Delete a Message from an Amazon SQS Queue*.

Send an Amazon SQS Message

You can use the Amazon SDK for .NET to send a message to an Amazon SQS queue.

Important: Due to the distributed nature of the queue, Amazon SQS cannot guarantee you will receive messages in the exact order they are sent. If you require that message order be preserved, place sequencing information in each message so you can reorder the messages upon receipt.

To send a message to an Amazon SQS queue

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_MESSAGE}";
   ```

   For more information about your queue URL, see *Amazon SQS Queue URLs*.

   Each queue message must be composed of only Unicode characters, and can be up to 64 kB in size. For more information about queue messages, go to SendMessage in the Amazon SQS service 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:

```
SendMessageResponse sendMessageResponse =
    amazonSQSClient.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 on deleting messages from your queue, see *Delete a Message from an Amazon SQS Queue*.

For information on receiving messages from your queue, see *Receive a Message from an Amazon SQS Queue*.

Receive a Message from an Amazon SQS Queue

You can use the Amazon SDK for .NET to receive messages from an Amazon SQS queue.

To receive a message from an Amazon SQS queue

1. Create and initialize a ReceiveMessageRequest instance. Specify the queue URL to receive a message from, as follows:

```
ReceiveMessageRequest 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:

```
ReceiveMessageResponse receiveMessageResponse =
  amazonSQSClient.ReceiveMessage(receiveMessageRequest);
```

The method returns a ReceiveMessageResponse instance, containing the list of messages the queue contains.

3. The response object contains a ReceiveMessageResult member. This member includes a Messages list. Iterate through this list to find a specific message, and use the `Body` property to determine if the list contains a specified message, as follows:

```
if (result.Message.Count != 0)
{
  for (int i = 0; i < result.Message.Count; i++)
  {
    if (result.Message[i].Body == messageBody)
    {
      receiptHandle = result.Message[i].ReceiptHandle;
    }
  }
}
```

Once the message is found in the list, use the `ReceiptHandle` property to obtain a receipt handle for the message. You can use this receipt handle to change message visibility timeout or to delete the

message from the queue. For more information about how to change the visibility timeout for a message, go to ChangeMessageVisibility.

For information about sending a message to your queue, see *Send an Amazon SQS Message*.

For more information about deleting a message from the queue, see *Delete a Message from an Amazon SQS Queue*.

Delete a Message from an Amazon SQS Queue

You can use the Amazon SDK for .NET to receive messages from an Amazon SQS queue.

To delete a message from an Amazon SQS queue

1. Create and initialize a DeleteMessageRequest instance. Specify the Amazon SQS queue to delete a message from and the receipt handle of the message to delete, as follows:

```
DeleteMessageRequest 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:

```
DeleteMessageResponse response =
  amazonSQSClient.DeleteMessage(deleteMessageRequest);
```

   Calling `DeleteMessage` unconditionally removes the message from the queue, regardless of the visibility timeout setting. For more information about visibility timeouts, go to 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*.

Related Resources

The following table lists related resources that you'll find useful when using Amazon SQS with the AWS SDK for .NET.

Resource	Description
Windows & .NET Developer Center	Provides sample code, documentation, tools, and additional resources to help you build applications on Amazon Web Services.
AWS SDK for .NET Documentation	Provides documentation for the AWS SDK for .NET.
Amazon Simple Queue Service (SQS) Documentation	Provides documentation for the Amazon SQS service.

4.11 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 the SDK, see the AWS SDK for .NET API Reference.

In addition to the namespaces for individual AWS services, the the SDK also provides the following APIs:

Area	Description	Resources
AWS Support	Programmatic access to AWS Support cases and Trusted Advisor features.	See Amazon.AWSSupport and Amazon.AWSSupport.Model.
General	Helper classes and enumerations.	See Amazon and Amazon.Util.

Other general programming information for the the SDK includes the following:

- Overriding Endpoints in the AWS SDK for .NET
- .NET Object Lifecycles

CHAPTER 5

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 AWS 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 API Reference.

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. 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 and the accompanying Toolkit for Visual Studio User Guide.

CHAPTER 6

Document History

The following table describes the important changes since the last release of the *AWS SDK for .NET Developer Guide*.

Last documentation update: July 28th, 2015

Change	Description	Release Date
New SDK version	Version 3 of the AWS SDK for .NET released. For more information, see the AWS SDK for .NET Developer Guide.	July 28th, 2015
New topic	A new topic was added about additional options for adding settings to `App.Config` and `Web.Config` files: *Configuration Files Reference for AWS SDK for .NET*.	February 5, 2015
New topic	A new topic was added about programming with Amazon DynamoDB: *Amazon DynamoDB Programming with Expressions by Using the AWS SDK for .NET*.	February 5, 2015
New topics	Three new topics were added about programming with the new AWS Resource APIs for .NET: *AWS CloudFormation Programming with the AWS SDK for .NET*, *Amazon Glacier Programming with the AWS SDK for .NET*, and *Amazon Simple Notification Service Programming with the AWS SDK for .NET*.	January 8, 2015
New topics	Two new topics were added about programming with the the SDK: *Amazon Simple Storage Service Programming with the AWS SDK for .NET* and *Programming Additional AWS Services with the AWS SDK for .NET*.	January 8, 2015
Revised topic	An existing topic was revised to add information about programming with the new AWS Resource APIs for .NET: *Amazon Simple Queue Service Programming with the AWS SDK for .NET*.	January 8, 2015
New topics	Two new topics were added about programming with the new AWS Resource APIs for .NET: *Programming with the AWS Resource APIs for .NET* and *AWS Identity and Access Management Code Examples with the AWS Resource APIs for .NET*.	December 16, 2014
Reorganized Table of Contents	The *Programming AWS Services with the AWS SDK for .NET* section of the Table of Contents was reorganized to group topics by service. As a result, four new topics were added: *AWS Identity and Access Management Programming with the AWS SDK for .NET*, *Amazon Elastic Compute Cloud Programming with the AWS SDK for .NET*, *Amazon Simple Queue Service Programming with the AWS SDK for .NET*, and *Amazon Route 53 Programming with the AWS SDK for .NET*.	December 16, 2014
New topics	Two new topics were added about programming with Amazon DynamoDB: *Amazon DynamoDB Programming with the AWS SDK for .NET* and *JSON Support in Amazon DynamoDB with the AWS SDK for .NET*.	December 3, 2014
Renamed topic	The *Examples and Tutorials* topic was renamed to *Programming AWS Services with the AWS SDK for .NET*.	December 3, 2014
Support for .NET SDK version 2	This guide has been modified to support the latest version of the AWS SDK for .NET.	November 8, 2013
New topic	This topic tracks recent changes to the *AWS SDK for .NET Developer Guide*. It is intended as a companion to the release notes.	September 9, 2013