
AWS Mobile SDK

Xamarin Developer Guide

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is the AWS Mobile SDK for .NET and Xamarin?	1
What's included in the AWS Mobile SDK for .NET and Xamarin?	1
Compatibility	1
IDEs	2
How do I get the AWS Mobile SDK for .NET and Xamarin?	2
About the AWS Mobile Services	2
Amazon Cognito Identity	2
Amazon Cognito Sync	2
Mobile Analytics	2
Dynamo DB	3
Amazon Simple Notification Service	3
Setting Up the AWS Mobile SDK for .NET and Xamarin	4
Prerequisites	4
Step 1: Obtain AWS Credentials	4
Step 2: Set Permissions	5
Step 3: Create a New Project	6
Windows	6
OS X	6
Step 4: Install the AWS Mobile SDK for .NET and Xamarin	6
Windows	6
Mac (OS X)	7
Step 5: Configure the AWS Mobile SDK for .NET and Xamarin	7
Set Logging	7
Set the Region Endpoint	8
Configure the HTTP Proxy Settings	8
Correct for Clock Skew	8
Next Steps	8
Getting Started with the AWS Mobile SDK for .NET and Xamarin	10
Store and Retrieve Files with Amazon S3	10
Project Setup	10
Initialize the S3 TransferUtility Client	11
Upload a File to Amazon S3	12
Download a File from Amazon S3	12
Sync User Data with Cognito Sync	12
Project Setup	10
Initialize the CognitoSyncManager	13
Syncing User Data	13
Store and Retrieve Data with DynamoDB	14
Project Setup	10
Initialize AmazonDynamoDBClient	16
Create a Class	16
Save an Item	16
Retrieve an Item	17
Update an Item	17
Delete an Item	17
Tracking App Usage Data with Amazon Mobile Analytics	17
Project Setup	10
Initialize MobileAnalyticsManager	18
Track Session Events	19
Receive Push Notifications using SNS (Xamarin iOS)	19
Project Setup	10
Create an SNS Client	21
Register Your Application for Remote Notifications	21
Send a Message from the SNS Console to Your Endpoint	22

Receive Push Notifications using SNS (Xamarin Android)	22
Project Setup	10
Create an SNS client	21
Register Your Application for Remote Notifications	21
Send a Message from the SNS Console to Your Endpoint	22
Amazon Cognito Identity	27
What is Amazon Cognito Identity?	27
Using a Public Provider to Authenticate Users	27
Using Developer Authenticated Identities	27
Amazon Cognito Sync	28
What is Amazon Cognito Sync?	28
Amazon Mobile Analytics	29
Key Concepts	29
Report Types	29
Project Setup	10
Prerequisites	4
Configure Mobile Analytics Settings	18
Integrating Mobile Analytics with Your Application	30
Create an App in the Mobile Analytics Console	18
Create a MobileAnalyticsManager Client	30
Record Monetization Events	31
Record Custom Events	31
Recording Sessions	32
Amazon Simple Storage Service (S3)	33
What is S3?	33
Key Concepts	29
Bucket	33
Objects	33
Object Metadata	33
Project Setup	10
Prerequisites	4
Create an S3 Bucket	34
Set Permissions for S3	10
(optional) Configure the Signature Version for S3 Requests	11
Integrating S3 with Your Application	35
Using the S3 Transfer Utility	35
Initialize the TransferUtility	35
(optional) Configure the TransferUtility	35
Download a File	36
Upload a File	36
Using the Service Level S3 APIs	36
Initialize the Amazon S3 Client	36
Download a File	36
Upload a File	36
Delete an Item	17
Delete Multiple Items	38
List Buckets	38
List Objects	39
Get a Bucket's Region	39
Get a Bucket's Policy	39
Amazon DynamoDB	41
What is Amazon DynamoDB?	41
Key Concepts	29
Tables	41
Items and Attributes	41
Data Types	41
Primary Key	42

Secondary Indexes	42
Query and Scan	42
Project Setup	10
Prerequisites	4
Create a DynamoDB Table	14
Set Permissions for DynamoDB	15
Integrating DynamoDB with your Application	44
Using the Document Model	44
Create a DynamoDB Client	45
CRUD Operations	45
Using the Object Persistence Model	47
Overview	47
Supported Data Types	47
Create a DynamoDB Client	45
CRUD Operations	45
Query and Scan	42
Using the DynamoDB Service Level APIs	50
Create a DynamoDB Client	45
CRUD Operations	45
Query and Scan	42
Amazon Simple Notification Service (SNS)	54
Key Concepts	29
Topics	54
Subscriptions	54
Publishing	54
Project Setup	10
Prerequisites	4
Integrating SNS with Your Application	55
Send Push Notifications (Xamarin Android)	55
Project Setup	10
Create an SNS client	21
Register Your Application for Remote Notifications	21
Send a Message from the SNS Console to Your Endpoint	22
Send Push Notifications (Xamarin iOS)	58
Project Setup	10
Create an SNS Client	21
Register Your Application for Remote Notifications	21
Send a Message from the SNS Console to Your Endpoint	22
Send and Receive SMS Notifications	61
Create a Topic	61
Subscribe to a Topic Using the SMS Protocol	62
Publish a Message	63
Send Messages to HTTP/HTTPS Endpoints	64
Configure Your HTTP/HTTPS Endpoint to Receive Amazon SNS Messages	64
Subscribe Your HTTP/HTTPS endpoint to Your Amazon SNS Topic	64
Confirm Your Subscription	64
Send Messages to the HTTP/HTTPS Endpoint	65
Troubleshooting SNS	65
Using Delivery Status in the Amazon SNS Console	65
Best Practices for Using the AWS Mobile SDK for .NET and Xamarin	66
Library of AWS Service Documentation	66
Amazon Cognito Identity	27
Amazon Cognito Sync	2
Amazon Mobile Analytics	29
Amazon S3	66
Amazon DynamoDB	67
Amazon Simply Notification Service (SNS)	67

Other Helpful Links	67
Troubleshooting	68
Ensure IAM Role Has Required Permissions	68
Using a HTTP Proxy Debugger	69
Document History	70

What is the AWS Mobile SDK for .NET and Xamarin?

The AWS Mobile SDK for .NET and Xamarin provides a set of .NET libraries, code samples, and documentation to help developers build connected mobile applications for:

- Xamarin iOS
- Xamarin Android
- Windows Phone Silverlight
- Windows RT 8.1
- Windows Phone 8.1

Mobile apps written using the AWS Mobile SDK for .NET and Xamarin call native platform APIs so they have the look and feel of native applications. The .NET libraries in the SDK provide C# wrappers around the AWS REST APIs.

What's included in the AWS Mobile SDK for .NET and Xamarin?

Supported AWS services currently include, but are not limited to:

- [Amazon Cognito](#)
- [Amazon S3](#)
- [Amazon DynamoDB](#)
- [Amazon Mobile Analytics](#)
- [Amazon Simple Notification Service](#)

These services enable you to authenticate users, save player and game data, save objects in the cloud, receive push notifications, and collect and analyze usage data.

The AWS Mobile SDK for .NET and Xamarin also allows you to use most of the AWS services supported by the AWS SDK for .NET. The AWS services specific to mobile development are explained in this developer guide. For more information about the AWS SDK for .NET, see:

- [AWS SDK for .NET Getting Started Guide](#)
- [AWS SDK for .NET Developer Guide](#)
- [AWS SDK for .NET API Reference](#)

Compatibility

The AWS Mobile SDK for .NET and Xamarin is shipped as a Portable Class Library (PCL). PCL Support was added in Xamarin.Android 4.10.1, Xamarin.iOS 7.0.4 and Xamarin Studio 4.2. Portable Library projects are automatically enabled in Xamarin Studio on OS X, and are built in to Visual Studio 2013.

IDEs

- **Windows:** You can use either Visual Studio or Xamarin Studio to develop your application.
- **Mac:** You must use the Xamarin Studio IDE to develop your applications. iOS development using Xamarin requires access to a Mac to run your app. For more information, see [Installing Xamarin.iOS on Windows](#).

How do I get the AWS Mobile SDK for .NET and Xamarin?

To get the AWS Mobile SDK for .NET and Xamarin, see [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#). The AWS Mobile SDK for .NET and Xamarin is distributed as NuGet packages. You can find a complete list of AWS service packages at [AWS SDK packages on NuGet](#) or at the [AWS SDK for .NET Github Repository](#).

About the AWS Mobile Services

Amazon Cognito Identity

All calls made to AWS require AWS credentials. Rather than hard-coding your credentials into your apps, we recommend that you use [Amazon Cognito Identity](#) to provide AWS credentials to your application. Follow the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to obtain AWS credentials via Amazon Cognito.

Cognito also allows you to authenticate users using public log-in providers like Amazon, Facebook, Twitter, and Google as well as providers that support [OpenID Connect](#). Cognito also works with unauthenticated users. Cognito provides temporary credentials with limited access rights that you specify with an [Identity and Access Management \(IAM\)](#) role. Cognito is configured by creating an identity pool that is associated with an IAM role. The IAM role specifies the resources/services your app may access.

To get started with Cognito Identity, see [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).

To learn more about Cognito Identity, see [Amazon Cognito Identity \(p. 27\)](#).

Amazon Cognito Sync

Cognito Sync is an AWS service and client library that enables cross-device syncing of application-related user data. You can use the Cognito Sync API to synchronize user profile data across devices and across login providers - Amazon, Facebook, Google, and your own custom identity provider.

To get started with Cognito Sync, see [Sync User Data with Cognito Sync \(p. 12\)](#).

For more information about Cognito Sync, see [Amazon Cognito Sync \(p. 28\)](#).

Mobile Analytics

Amazon Mobile Analytics lets you collect, visualize, and understand app usage for your mobile apps. Reports are available for metrics on active users, sessions, retention, in-app revenue, and custom events,

and can be filtered by platform and date range. Amazon Mobile Analytics is built to scale with your business and can collect and process billions of events from many millions of endpoints.

To get started using Mobile Analytics, see [Tracking App Usage Data with Amazon Mobile Analytics \(p. 17\)](#).

For more information about Mobile Analytics, see [Amazon Mobile Analytics \(p. 29\)](#).

Dynamo DB

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

To get started using Dynamo DB, see [Store and Retrieve Data with DynamoDB \(p. 14\)](#).

For more information about Dynamo DB, see [Amazon DynamoDB \(p. 41\)](#).

Amazon Simple Notification Service

Amazon Simple Notification Service (SNS) is a fast, flexible, fully managed push notification service that lets you send individual messages or to fan-out messages to large numbers of recipients. Amazon Simple Notification Service makes it simple and cost effective to send push notifications to mobile device users, email recipients or even send messages to other distributed services.

To get started using SNS for Xamarin iOS, see [Receive Push Notifications using SNS \(Xamarin iOS\) \(p. 19\)](#).

To get started using SNS for Xamarin Android, see [Receive Push Notifications using SNS \(Xamarin Android\) \(p. 22\)](#).

For more information about SNS, see [Amazon Simple Notification Service \(SNS\) \(p. 54\)](#).

Setting Up the AWS Mobile SDK for .NET and Xamarin

You can set up the AWS Mobile SDK for .NET and Xamarin and start building a new project or you can integrate the SDK with an existing project. You can also clone and run the [samples](#) to get a sense of how the SDK works. Follow these steps to set up and start using the AWS Mobile SDK for .NET and Xamarin.

Prerequisites

Before you can use the AWS Mobile SDK for .NET and Xamarin, you must do the following:

- Create an [An AWS Account](#).
- Install the [Xamarin platform](#).
- **Windows users:** complete [Xamarin: Getting Started for Windows](#)
- **Mac users:** complete [Xamarin: Getting Started for Mac](#).

After you complete the prerequisites:

1. Obtain AWS credentials by using Amazon Cognito.
2. Set the required permissions for each AWS service that you will use in your application.
3. Create a new project in your IDE.
4. Install the AWS Mobile SDK for .NET and Xamarin.
5. Configure the AWS Mobile SDK for .NET and Xamarin.

Step 1: Obtain AWS Credentials

To make calls to AWS in your application, you must first obtain AWS credentials. You do this by using Amazon Cognito, an AWS service that allows your application to access the services in the SDK without having to embed your private AWS credentials in the application.

To get started with Amazon Cognito, you need to create an identity pool. An identity pool is a store of information that is specific to your account and is identified by a unique identity pool ID that looks like the following.:

```
"us-east-1:00000000-0000-0000-0000-000000000000"
```

1. Log in to the [Amazon Cognito Console](#), choose **Manage Federated Identities**, and then choose **Create new identity pool**.
2. Enter a name for your identity pool and select the checkbox to enable access to unauthenticated identities. Choose **Create Pool** to create your identity pool.
3. Choose **Allow** to create the two default roles associated with your identity pool, one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Amazon Cognito Sync and Amazon Mobile Analytics.

Typically, you will only use one identity pool per application.

After you create your identity pool, you obtain AWS credentials by creating a `CognitoAWSCredentials` object (passing it your identity pool ID) and then passing it to the constructor of an AWS client as follows.:

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials (
    "us-east-1:00000000-0000-0000-0000-000000000000", // Your identity pool ID
    RegionEndpoint.USEast1 // Region
);

// Example for |MA|
analyticsManager = MobileAnalyticsManager.GetOrCreateInstance(
    credentials,
    RegionEndpoint.USEast1, // Region
    APP_ID // app id
);
```

Step 2: Set Permissions

You need to set permissions for every AWS service that you want to use in your application. First, you need to understand how AWS views the users of your application.

When someone uses your application and makes calls to AWS, AWS assigns that user an identity. The identity pool that you created in Step 1 is where AWS stores these identities. There are two types of identities: authenticated and unauthenticated. Authenticated identities belong to users who are authenticated by a public login provider (e.g., Facebook, Amazon, Google). Unauthenticated identities belong to guest users.

Every identity is associated with an AWS Identity and Access Management role. In Step 1, you created two IAM roles, one for authenticated users and one for unauthenticated users. Every IAM role has one or more policies attached to it that specify which AWS services the identities assigned to that role can access. For example, the following sample policy grants access to an Amazon S3 bucket.:

```
{
  "Statement": [
    {
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::MYBUCKETNAME/*",
      "Principal": "*"
    }
  ]
}
```

To set permissions for the AWS services that you want to use in your application, simply modify the policy attached to the roles.

1. Go to the [IAM Console and choose Roles](#). Type your identity pool name into the search box. Choose the IAM role that you want to configure. If your application allows both authenticated and unauthenticated users, you need to grant permissions for both roles.
2. Click **Attach Policy**, select the policy you want, and then click **Attach Policy**. The default policies for the IAM roles that you created provide access to Amazon Cognito Sync and Mobile Analytics.

For more information about creating policies or to choose from a list of existing policies, see [IAM Policies](#).

Step 3: Create a New Project

Windows

You can use either Visual Studio or Xamarin Studio to develop your application.

OS X

You must use the Xamarin Studio IDE to develop your applications. iOS development using Xamarin requires access to a Mac to run your app. For more information, see [Installing Xamarin.iOS on Windows](#).

Step 4: Install the AWS Mobile SDK for .NET and Xamarin

Windows

Option 1: Install by Using the Package Manager Console

The AWS Mobile SDK for .NET and Xamarin consists a set of .NET assemblies. To install the AWS Mobile SDK for .NET and Xamarin, run the install-package command for each package in the Package Manager console. For example, to install Cognito Identity, run the following.:

```
Install-Package AWSSDK.CognitoIdentity
```

The AWS Core Runtime and Amazon Cognito Identity packages are required for all projects. The following is a full list of package names for each service.

Service	Package name
AWS Core Runtime	AWSSDK.Core
Amazon Cognito Sync	AWSSDK.CognitoSync
Amazon Cognito Identity	AWSSDK.CognitoIdentity
Amazon DynamoDB	AWSSDK.DynamoDBv2
Amazon Mobile Analytics	AWSSDK.MobileAnalytics
Amazon S3	AWSSDK.S3
Amazon SNS	AWSSDK.SimpleNotificationService

To include a prerelease package, include the `-Pre` command line argument while installing the package as follows.:

```
Install-Package AWSSDK.CognitoSync -Pre
```

You can find a complete list of AWS service packages at [AWS SDK packages on NuGet](#) or at the [AWS SDK for .NET Github Repository](#).

Option 2: Install by Using Your IDE

In Visual Studio

1. Right-click the project, and then click **Manage NuGet Packages**.
2. Search for the package name that you want to add to your project. To include the prelease NuGet packages, choose **Include Prelease**. You can find a complete list of AWS service packages at [AWS SDK packages on NuGet](#).
3. Choose the package, and then choose **Install**.

In Xamarin Studio

1. Right-click the packages folder, and then choose **Add Packages**.
2. Search for the package name that you want to add to your project. To include the prelease NuGet packages, choose **Show pre-release packages**. You can find a complete list of AWS service packages at [AWS SDK packages on NuGet](#).
3. Select the checkbox next to the package you want, and then choose **Add Package**.

Mac (OS X)

In Xamarin Studio

1. Right-click the packages folder, and then choose **Add Packages**.
2. Search for the package name that you want to add to your project. To include the prelease NuGet packages, choose **Show pre-release packages**. You can find a complete list of AWS service packages at [AWS SDK packages on NuGet](#).
3. Select the checkbox next to the package you want, and then choose **Add Package**.

Important

If you are developing using a Portable Class Library, you must also add the AWSSDK.Core NuGet package to all projects deriving from the Portable Class Library.

Step 5: Configure the AWS Mobile SDK for .NET and Xamarin

Set Logging

You set logging settings by using the `Amazon.AWSConfigs` class and the `Amazon.Util.LoggingConfig` class. You can find these in the `AWSSdk.Core` assembly, available through the Nuget Package Manager in Visual Studio. You can place the logging settings code in the `OnCreate` method in the `MainActivity.cs` file for Android apps or the `AppDelegate.cs` file for iOS apps. You should also add `using Amazon` and `using Amazon.Util` statements to the `.cs` files.

Configure logging settings as follows.:

```
var loggingConfig = AWSConfigs.LoggingConfig;
```

```
loggingConfig.LogMetrics = true;  
loggingConfig.LogResponses = ResponseLoggingOption.Always;  
loggingConfig.LogMetricsFormat = LogMetricsFormatOption.JSON;  
loggingConfig.LogTo = LoggingOptions.SystemDiagnostics;
```

When you log to SystemDiagnostics, the framework internally prints the output to the System.Console. If you want to log HTTP responses, set the LogResponses flag. The values can be Always, Never, or OnError.

You can also log performance metrics for HTTP requests by using the LogMetrics property. The log format can be specified by using LogMetricsFormat property. Valid values are JSON or standard.

Set the Region Endpoint

Configure the default region for all service clients as follows.:

```
AWSConfigs.AWSRegion="us-east-1";
```

This sets the default region for all the service clients in the SDK. You can override this setting by explicitly specifying the region at the time of creating an instance of the service client, as follows.:

```
IAmazonS3 s3Client = new AmazonS3Client(credentials,RegionEndpoint.USEast1);
```

Configure the HTTP Proxy Settings

If your network is behind a proxy, you can configure the proxy settings for the HTTP requests as follows.

```
var proxyConfig = AWSConfigs.ProxyConfig;  
proxyConfig.Host = "localhost";  
proxyConfig.Port = 80;  
proxyConfig.Username = "<username>";  
proxyConfig.Password = "<password>";
```

Correct for Clock Skew

This property determines if the SDK should correct for client clock skew by determining the correct server time and reissuing the request with the correct time.

```
AWSConfigs.CorrectForClockSkew = true;
```

This field is set if a service call resulted in an exception and the SDK has determined that there is a difference between local and server times.

```
var offset = AWSConfigs.ClockOffset;
```

To learn more about clock skew, see [Clock-skew Correction](#) on the AWS Blog.

Next Steps

Now that you have set up the AWS Mobile SDK for .NET and Xamarin, you can:

- **Get started.** Read [Getting Started with the AWS Mobile SDK for .NET and Xamarin \(p. 10\)](#) for quick-start instructions about how to use and configure the services in the AWS Mobile SDK for .NET and Xamarin.
- **Explore the service topics.** Learn about each service and how it works in the AWS Mobile SDK for .NET and Xamarin.
- **Run the demos.** View our [sample Xamarin applications](#) that demonstrate common use cases. To run the sample apps, set up the AWS Mobile SDK for .NET and Xamarin as described previously, and then follow the instructions contained in the README files of the individual samples.
- **Learn the APIs.** View the [sdk-xamarin-ref](#).
- **Ask questions:** Post questions on the [AWS Mobile SDK Forums](#) or [open an issue on Github](#).

Getting Started with the AWS Mobile SDK for .NET and Xamarin

The AWS Mobile SDK for .NET and Xamarin provides the libraries, samples and, documentation needed to call AWS services from Xamarin applications.

You must complete all of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before you start using the services below.

These getting started topics will walk you through the following:

Topics

- [Store and Retrieve Files with Amazon S3 \(p. 10\)](#)
- [Sync User Data with Cognito Sync \(p. 12\)](#)
- [Store and Retrieve Data with DynamoDB \(p. 14\)](#)
- [Tracking App Usage Data with Amazon Mobile Analytics \(p. 17\)](#)
- [Receive Push Notifications using SNS \(Xamarin iOS\) \(p. 19\)](#)
- [Receive Push Notifications using SNS \(Xamarin Android\) \(p. 22\)](#)

For information about other AWS Mobile SDKs, see [AWS Mobile SDK](#).

Store and Retrieve Files with Amazon S3

Amazon Simple Storage Service (Amazon S3) provides mobile developers with secure, durable, highly-scalable object storage. Amazon S3 is easy to use, with a simple web services interface to store and retrieve any amount of data from anywhere on the web.

The tutorial below explains how to integrate the S3 TransferUtility, a high-level utility for using S3 with your app. For more information about using S3 from Xamarin applications, see [Amazon Simple Storage Service \(S3\) \(p. 33\)](#).

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

This tutorial also assumes you have already created an S3 bucket. To create an S3 bucket, visit the [S3 AWS Console](#).

Set Permissions for S3

The default IAM role policy grants your application access to Amazon Mobile Analytics and Amazon Cognito Sync. In order for your Cognito identity pool to access Amazon S3, you must modify the identity pool's roles.

1. Go to the [Identity and Access Management Console](#) and click **Roles** in the left-hand pane.
2. Type your identity pool name into the search box. Two roles will be listed: one for unauthenticated users and one for authenticated users.

3. Click the role for unauthenticated users (it will have unauth appended to your identity pool name).
4. Click **Create Role Policy**, select **Policy Generator**, and then click **Select**.
5. On the **Edit Permissions** page, enter the settings shown in the following image, replacing the Amazon Resource Name (ARN) with your own. The ARN of an S3 bucket looks like `arn:aws:s3:::examplebucket/*` and is composed of the region in which the bucket is located and the name of the bucket. The settings shown below will give your identity pool full to access to all actions for the specified bucket.

Edit Permissions

The policy generator enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see [Overview of Policies](#) in Using AWS Identity and Access Management.

Effect: Allow ☒ Deny ☐

AWS Service: Amazon S3

Actions: All Actions Selected

Amazon Resource Name (ARN): arn:aws:s3:::examplebucket/*

[Add Conditions \(optional\)](#)

Add Statement

1. Click the **Add Statement** button and then click **Next Step**.
2. The Wizard will show you the configuration that you generated. Click **Apply Policy**.

For more information on granting access to S3, see [Granting Access to an Amazon S3 Bucket](#).

Add NuGet Package for S3 to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the S3 NuGet package to your project.

(optional) Configure the Signature Version for S3 Requests

Every interaction with Amazon S3 is either authenticated or anonymous. AWS uses the Signature Version 4 or Signature Version 2 algorithms to authenticate calls to the service.

All new AWS regions created after January 2014 only support Signature Version 4. However, many older regions still support Signature Version 4 and Signature Version 2 requests.

If your bucket is in one of the regions that does not support Signature Version 2 requests as listed on [this page](#), you must set the `AWSConfigsS3.UseSignatureVersion4` property to "true" like so:

```
AWSConfigsS3.UseSignatureVersion4 = true;
```

For more information on AWS Signature versions, see [Authenticating Requests \(AWS Signature Version 4\)](#).

Initialize the S3 TransferUtility Client

Create an S3 client, passing it your AWS credentials object, and then pass the S3 client to the transfer utility, like so:

```
var s3Client = new AmazonS3Client(credentials,region);
```

```
var transferUtility = new TransferUtility(s3Client);
```

Upload a File to Amazon S3

To upload a file to S3, call `Upload` on the Transfer Utility object, passing the following parameters:

- `file` - String name of the file that you want to upload
- `bucketName` - String name of the S3 bucket to store the file

```
transferUtility.Upload(  
    Path.Combine(Environment.SpecialFolder.ApplicationData, "file"),  
    "bucketName"  
);
```

The code above assumes that there is a file in the directory `Environment.SpecialFolder.ApplicationData`. Uploads automatically use S3's multi-part upload functionality on large files to enhance throughput.

Download a File from Amazon S3

To download a file from S3, call `Download` on the Transfer Utility object, passing the following parameters:

- `file` - String name of the file that you want to download
- `bucketName` - String name of the S3 bucket from which you want to download the file
- `key` - A string representing the name of the S3 object (a file in this case) to download

```
transferUtility.Download(  
    Path.Combine(Environment.SpecialFolder.ApplicationData, "file"),  
    "bucketName",  
    "key"  
);
```

For more information on accessing Amazon S3 from an Xamarin application, see [Amazon Simple Storage Service \(S3\) \(p. 33\)](#).

Sync User Data with Cognito Sync

Amazon Cognito Sync makes it easy to save mobile user data, such as app preferences or game state in the AWS Cloud without writing any backend code or managing any infrastructure. You can save data locally on users' devices allowing your applications to work even when the devices are offline. You can also synchronize data across a user's devices so that their app experience will be consistent regardless of the device they use.

The tutorial below explains how to integrate Sync with your app.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Grant Access to Your Cognito Sync Resources

The default policy associated with the unauthenticated and authenticate roles that you created during setup grant your application access to Cognito Sync. No further configuration is required.

Add NuGet Package for Cognito Sync to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Cognito SyncManager NuGet package to your project.

Initialize the CognitoSyncManager

Pass your initialized Amazon Cognito credentials provider to the `CognitoSyncManager` constructor:

```
CognitoSyncManager syncManager = new CognitoSyncManager (
    credentials,
    new AmazonCognitoSyncConfig {
        RegionEndpoint = RegionEndpoint.USEast1 // Region
    }
);
```

Syncing User Data

To sync unauthenticated user data:

1. Create a dataset.
2. Add user data to the dataset.
3. Synchronize the dataset with the cloud.

Create a Dataset

Create an instance of `Dataset`. The `openOrCreateDataset` method is used to create a new dataset or open an existing instance of a dataset stored locally on the device:

```
Dataset dataset = syncManager.OpenOrCreateDataset("myDataset");
```

Add User Data to the Dataset

User data is added in the form of key/value pairs:

```
dataset.OnSyncSuccess += SyncSuccessCallback;
dataset.Put("myKey", "myValue");
```

Cognito datasets function as dictionaries, with values accessible by key:

```
string myValue = dataset.Get("myKey");
```

Synchronize Dataset

To synchronize a dataset, call its `synchronize` method:

```
dataset.SynchronizeAsync();

void SyncSuccessCallback(object sender, SyncSuccessEventArgs e) {
    // Your handler code here
}
```

All data written to datasets will be stored locally until the dataset is synced. The code in this section assumes you are using an unauthenticated Cognito identity, so when the user data is synced with the cloud it will be stored per device. The device has a device ID associated with it. When the user data is synced to the cloud, it will be associated with that device ID.

For more information on Cognito Sync, see [Amazon Cognito Sync \(p. 28\)](#).

Store and Retrieve Data with DynamoDB

[Amazon DynamoDB](#) is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

The tutorial below explains how to integrate the DynamoDB Object Persistence Model with your app, which stores objects in DynamoDB.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Create a DynamoDB Table

Before you can read and write data to a DynamoDB database, you must create a table. When creating a table you must specify the primary key. The primary key is composed of a hash attribute and an optional range attribute. For more information on how primary and range attributes are used, see [Working With Tables](#).

1. Go to the [DynamoDB Console](#) and click **Create Table**. The Create Table wizard appears.
2. Specify your table name, primary key type (Hash), and hash attribute name ("Id") as shown below, and then click **Continue**:

Create Table Cancel

PRIMARY KEY | ADD INDEXES (optional) | PROVISIONED THROUGHPUT CAPACITY | ADDITIONAL OPTIONS (optional) | SUMMARY

Table Name: Books
Table will be created in us-east-1 region

Primary Key:
DynamoDB is a schema-less database. You only need to tell us your primary key attribute(s).

Primary Key Type: ☐ Hash and Range ☒ Hash

Hash Attribute Name: ☐ String ☐ Number ☐ Binary
Id

Warning: Choose a hash attribute that ensures that your workload is evenly distributed across hash keys.
For example, "Customer ID" is a good hash key, while "Game ID" would be a bad choice if most of your traffic relates to a few popular games.
[Learn more about choosing your primary key](#)

Cancel Continue Help

3. Leave the edit fields in the next screen empty and click **Continue**.
4. Accept the default values for **Read Capacity Units** and **Write Capacity Units** and click **Continue**.
5. On the next screen enter your email address in the **Send notification to:** text box and click **Continue**. The review screen appears.
6. Click **Create**. It may take a few minutes for your table to be created.

Set Permissions for DynamoDB

In order for your identity pool to access Amazon DynamoDB, you must modify the identity pool's roles.

1. Navigate to the [Identity and Access Management Console](#) and click **Roles** in the left-hand pane. Search for your identity pool name - two roles will be listed one for unauthenticated users and one for authenticated users.
2. Click the role for unauthenticated users (it will have "unauth" appended to your identity pool name) and click **Create Role Policy**.
3. Select **Policy Generator** and click **Select**.
4. On the **Edit Permissions** page, enter the settings shown in the following image. The Amazon Resource Name (ARN) of a DynamoDB table looks like `arn:aws:dynamodb:us-west-2:123456789012:table/Books` and is composed of the region in which the table is located, the owner's AWS account number, and the name of the table in the format `table/Books`. For more information about specifying ARNs, see [Amazon Resource Names for DynamoDB](#).

Edit Permissions

The policy generator enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see [Overview of Policies](#) in Using AWS Identity and Access Management.

Effect: Allow ☒ Deny ☐

AWS Service: Amazon DynamoDB

Actions: All Actions Selected

Amazon Resource Name (ARN): arn:aws:dynamodb:us-west-2:1:

[Add Conditions \(optional\)](#)

Add Statement

5. Click **Add Statement**, and then click **Next Step**. The Wizard will show you the configuration generated.
6. Click **Apply Policy**.

Add NuGet package for DynamoDB to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the DynamoDB NuGet package to your project.

Initialize AmazonDynamoDBClient

Pass your initialized Amazon Cognito credentials provider and your region to the `AmazonDynamoDB` constructor, then pass the client to the `DynamoDBContext`:

```
var client = new AmazonDynamoDBClient(credentials, region);  
DynamoDBContext context = new DynamoDBContext(client);
```

Create a Class

To write a row to the table, define a class to hold your row data. The class should also contain properties that hold the attribute data for the row and will be mapped to the DynamoDB Table created in the console. The following class declaration illustrates such a class:

```
[DynamoDBTable("Books")]  
public class Book  
{  
    [DynamoDBHashKey]    // Hash key.  
    public int Id { get; set; }  
    public string Title { get; set; }  
    public string ISBN { get; set; }  
    public int Price { get; set; }  
    public string PageCount { get; set; }  
    public string Author { get; set; }  
}
```

Save an Item

To save an item, first create an object:

```
Book songOfIceAndFire = new Book()
```

```
{  
    Id=1,  
    Title="Game Of Thrones",  
    ISBN="978-0553593716",  
    Price=4,  
    PageCount="819",  
    Author="GRRM"  
};
```

Then save it:

```
context.Save(songOfIceAndFire);
```

To update a row, modify the instance of the `DDTableRow` class and call `AWSDynamoObjectMapper.save()` as shown above.

Retrieve an Item

Retrieve an item using a primary key:

```
Book retrievedBook = context.Load<Book>(1);
```

Update an Item

To update an item:

```
Book retrievedBook = context.Load<Book>(1);  
retrievedBook.ISBN = "978-0553593716";  
context.Save(retrievedBook);
```

Delete an Item

To delete an item:

```
Book retrievedBook = context.Load<Book>(1);  
context.Delete(retrievedBook);
```

For more information on accessing DynamoDB from a Xamarin application, see [Amazon DynamoDB \(p. 41\)](#).

Tracking App Usage Data with Amazon Mobile Analytics

Amazon Mobile Analytics allows you to measure app usage and app revenue. By tracking key trends such as new vs. returning users, app revenue, user retention, and custom in-app behavior events, you can make data-driven decisions to increase engagement and monetization for your app.

The tutorial below explains how to integrate Mobile Analytics with your app.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Create an App in the Mobile Analytics Console

Go to the [Amazon Mobile Analytics console](#) and create an app. Note the `appId` value, as you'll need it later. When you are creating an app in the Mobile Analytics Console you will need to specify your identity pool ID. For instructions on creating an identity pool, see [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).

To learn more about working in the console, see the [Amazon Mobile Analytics User Guide](#).

Set Permissions for Mobile Analytics

The default policy associated with the roles that you created during setup grant your application access to Mobile Analytics. No further configuration is required.

Add NuGet Package for Mobile Analytics to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Mobile Analytics NuGet package to your project.

Configure Mobile Analytics Settings

Mobile Analytics defines some settings that can be configured in the `awsconfig.xml` file:

```
var config = new MobileAnalyticsManagerConfig();
config.AllowUseDataNetwork = true;
config.DBWarningThreshold = 0.9f;
config.MaxDBSize = 5242880;
config.MaxRequestSize = 102400;
config.SessionTimeout = 5;
```

- `AllowUseDataNetwork` - A boolean that specifies if the session events are sent on the data network.
- `DBWarningThreshold` - This is the limit on the size of the database which, once reached, will generate warning logs.
- `MaxDBSize` - This is the size of the SQLite Database. When the database reaches the maximum size, any additional events are dropped.
- `MaxRequestSize` - This is the maximum size of the request in Bytes that should be transmitted in an HTTP request to the mobile analytics service.
- `SessionTimeout` - This the time interval after an application goes to background and when session can be terminated.

The settings shown above are the default values for each configuration item.

Initialize MobileAnalyticsManager

To initialize your `MobileAnalyticsManager`, call `GetOrCreateInstance` on your `MobileAnalyticsManager`, passing in your AWS credentials, your region, your Mobile Analytics application ID, and your optional config object:


```
var manager = MobileAnalyticsManager.GetOrCreateInstance(  
    "APP_ID",  
    "Credentials",  
    "RegionEndPoint",  
    config  
);
```

Track Session Events

Xamarin Android

Override the activity's `OnPause()` and `OnResume()` methods to record session events.

```
protected override void OnResume()  
{  
    manager.ResumeSession();  
    base.OnResume();  
}  
  
protected override void OnPause()  
{  
    manager.PauseSession();  
    base.OnPause();  
}
```

This needs to be implemented for each activity in your application.

Xamarin iOS

In your `AppDelegate.cs`:

```
public override void DidEnterBackground(UIApplication application)  
{  
    manager.PauseSession();  
}  
  
public override void WillEnterForeground(UIApplication application)  
{  
    manager.ResumeSession();  
}
```

For more information on Mobile Analytics, see [Amazon Mobile Analytics \(p. 29\)](#).

Receive Push Notifications using SNS (Xamarin iOS)

This document explains how to send push notifications to a Xamarin iOS application using Amazon Simple Notification Service (SNS) and the AWS Mobile SDK for .NET and Xamarin.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Set Permissions for SNS

Follow Step 2 in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to attach the policy mentioned below to your application's roles. This will give your application the proper permissions to access SNS:

1. Go to the [IAM Console](#) and select the IAM role that you want to configure.
2. Click **Attach Policy**, select the AmazonSNSFullAccess policy and click **Attach Policy**.

Warning

Using AmazonSNSFullAccess is not recommended in a production environment. We use it here to allow you to get up and running quickly. For more information about specifying permissions for an IAM role, see [Overview of IAM Role Permissions](#).

Obtain Membership in the Apple iOS Developer Program

You will need to run your app on a physical device to receive push notifications. To run your app on a device, you must have a membership in the [Apple iOS Developer Program Membership](#). Once you have a membership, you can use Xcode to generate a signing identity. For more information, see Apple's [App Distribution Quick Start](#) documentation.

Create an iOS Certificate

First, you need to create an iOS Certificate. Then, you need to create a provisioning profile configured for push notifications. To do so:

1. Go to the [Apple Developer Member Center](#), click **Certificates, Identifiers & Profiles**.
2. Click **Identifiers** under **iOS Apps**, click the plus button in the upper right-hand corner of the web page to add a new iOS App ID, and enter an App ID description.
3. Scroll down to the **Add ID Suffix** section and select **Explicit App ID** and enter your bundle identifier.
4. Scroll down to the **App Services** section and select **Push Notifications**.
5. Click **Continue**.
6. Click **Submit**.
7. Click **Done**.
8. Select the App ID you just created and then click **Edit**.
9. Scroll down to the **Push Notifications** section. Click **Create Certificate** under **Development SSL Certificate**.
- 10 Follow the instructions to create a Certificate Signing Request (CSR), upload the request, and download an SSL certificate that will be used to communicate with Apple Notification Service (APNS).
- 11 Return to the **Certificates, Identifiers & Profiles** page. Click **All** under **Provisioning Profiles**.
- 12 Click the plus button in the upper right-hand corner to add a new provisioning profile.
- 13 Select **iOS App Development**, and then click **Continue**.
- 14 Select your App ID, and then click **Continue**.
- 15 Select your developer certificate, and then click **Continue**.

- 16 Select your device, and then click **Continue**.
- 17 Enter a profile name, and then click **Generate**.
- 18 Download and double-click the provision file to install the provisioning profile.

For more information about provisioning a profile configured for push notifications, see Apple's [Configuring Push Notifications](#) documentation.

Use Certificate to Create Platform ARN in SNS Console

1. Run the KeyChain access app, select **My Certificates** on the lower left-hand side of the screen, and then right-click the SSL certificate you generated to connect to APNS and select **Export**. You will be prompted to specify a name for the file and a password to protect the certificate. The certificate will be saved in a P12 file.
2. Go to the [SNS Console](#) and click **Applications** on the left-hand side of the screen.
3. Click **Create platform application** to create a new SNS platform application.
4. Enter an **Application Name**.
5. Select **Apple Development** for **Push notification platform**.
6. Click **Choose File** and select the P12 file you created when you exported your SSL certificate.
7. Enter the password you specified when you exported the SSL certificate and click **Load Credentials From File**.
8. Click **Create platform application**.
9. Select the Platform Application you just created and copy the Application ARN. You will need this in the upcoming steps.

Add NuGet Package for SNS to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Amazon Simple Notification Service NuGet package to your project.

Create an SNS Client

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

Register Your Application for Remote Notifications

To register an application, call `RegisterForRemoteNotifications` on your `UIApplication` object, as shown below. Place the following code in `AppDelegate.cs`, inserting your platform application ARN where prompted below:

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options) {  
    // do something  
    var pushSettings = UIUserNotificationSettings.GetSettingsForTypes (  
        UIUserNotificationType.Alert |  
        UIUserNotificationType.Badge |  
        UIUserNotificationType.Sound,  
        null  
    );  
    app.RegisterUserNotifications(pushSettings);  
    app.RegisterForRemoteNotifications();  
    // do something  
    return true;  
}
```

```
public override void RegisteredForRemoteNotifications(UIApplication application, NSData token) {
    var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ", "");
    if (!string.IsNullOrEmpty(deviceToken)) {
        //register with SNS to create an endpoint ARN
        var response = await SnsClient.CreatePlatformEndpointAsync(
            new CreatePlatformEndpointRequest {
                Token = deviceToken,
                PlatformApplicationArn = "YourPlatformArn" /* insert your platform application ARN
here */
            });
    }
}
```

Send a Message from the SNS Console to Your Endpoint

1. Go to the [SNS Console > Applications](#).
2. Select your platform application, select an endpoint, and click **Publish to endpoint**.
3. Type in a text message in the text box and click **Publish message** to publish a message.

Receive Push Notifications using SNS (Xamarin Android)

The tutorial explains how to send push notifications to a Xamarin Android application using Amazon Simple Notification Service (SNS) and the AWS Mobile SDK for .NET and Xamarin.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Set Permissions for SNS

Follow Step 2 in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to attach the policy mentioned below to your application's roles. This will give your application the proper permissions to access SNS:

1. Go to the [IAM Console](#) and select the IAM role that you want to configure.
2. Click **Attach Policy**, select the AmazonSNSFullAccess policy and click **Attach Policy**.

Warning

Using AmazonSNSFullAccess is not recommended in a production environment. We use it here to allow you to get up and running quickly. For more information about specifying permissions for an IAM role, see [Overview of IAM Role Permissions](#).

Enable Push Notifications on Google Cloud

First, add a new Google API project:

1. Go to the [Google Developers Console](#).
2. Click **Create Project**.
3. In the **New Project** box, enter a project name, take note of the project ID (you will need it later) and click **Create**.

Next, enable the Google Cloud Messaging (GCM) service for your project:

1. In the [Google Developers Console](#), your new project should already be selected. If not, select it in the drop-down at the top of the page.
2. Select **APIs & auth** from the side bar on the left-hand side of the page.
3. In the search box, type "Google Cloud Messaging for Android" and click the **Google Cloud Messaging for Android** link.
4. Click **Enable API**.

Finally, obtain an API Key:

1. In the Google Developers Console, select **APIs & auth > Credentials**.
2. Under **Public API access**, click **Create new key**.
3. In the **Create a new key** dialog, click **Server key**.
4. In the resulting dialog, click **Create** and copy the API key displayed. You will use this API key to perform authentication later on.

Use Project ID to Create a Platform ARN in SNS Console

1. Go to the [SNS Console](#).
2. Click **Applications** on the left-hand side of the screen.
3. Click **Create platform application** to create a new SNS platform application.
4. Enter an **Application Name**.
5. Select **Google Cloud Messaging (GCM)** for **Push notification platform**.
6. Paste the API key into the text box labeled **API key**.
7. Click **Create platform application**.
8. Select the Platform Application you just created and copy the Application ARN.

Add NuGet Package for SNS to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Amazon Simple Notification Service NuGet package to your project.

Create an SNS client

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

Register Your Application for Remote Notifications

In order to register for remote notifications on Android, you will need to create a `BroadcastReceiver` which can receive Google Cloud messages. Change the package name below where prompted to do so:

```
[BroadcastReceiver(Permission = "com.google.android.c2dm.permission.SEND")]
```

```
[IntentFilter(new string[] {
    "com.google.android.c2dm.intent.RECEIVE"
}, Categories = new string[] {
    "com.amazonaws.sns" /* change to match your package */
})]
[IntentFilter(new string[] {
    "com.google.android.c2dm.intent.REGISTRATION"
}, Categories = new string[] {
    "com.amazonaws.sns" /* change to match your package */
})]
[IntentFilter(new string[] {
    "com.google.android.gcm.intent.RETRY"
}, Categories = new string[] {
    "com.amazonaws.sns" /* change to match your package */
})]
public class GCMBroadcastReceiver: BroadcastReceiver {
    const string TAG = "PushHandlerBroadcastReceiver";
    public override void OnReceive(Context context, Intent intent) {
        GCMIntentService.RunIntentInService(context, intent);
        SetResult(Result.Ok, null, null);
    }
}

[BroadcastReceiver]
[IntentFilter(new[] {
    Android.Content.Intent.ActionBootCompleted
})]
public class GCMBootReceiver: BroadcastReceiver {
    public override void OnReceive(Context context, Intent intent) {
        GCMIntentService.RunIntentInService(context, intent);
        SetResult(Result.Ok, null, null);
    }
}
```

Below is the service that receives the push notification from the BroadcastReceiver and displays the notification on the device's notification bar:

```
[Service]
public class GCMIntentService: IntentService {
    static PowerManager.WakeLock sWakeLock;
    static object LOCK = new object();

    public static void RunIntentInService(Context context, Intent intent) {
        lock(LOCK) {
            if (sWakeLock == null) {
                // This is called from BroadcastReceiver, there is no init.
                var pm = PowerManager.FromContext(context);
                sWakeLock = pm.NewWakeLock(
                    WakeLockFlags.Partial, "My WakeLock Tag");
            }
        }

        sWakeLock.Acquire();
        intent.SetClass(context, typeof(GCMIntentService));
        context.StartService(intent);
    }

    protected override void OnHandleIntent(Intent intent) {
        try {
            Context context = this.ApplicationContext;
            string action = intent.Action;

            if (action.Equals("com.google.android.c2dm.intent.REGISTRATION")) {
                HandleRegistration(intent);
            }
        }
    }
}
```

```
        } else if (action.Equals("com.google.android.c2dm.intent.RECEIVE")) {  
            HandleMessage(intent);  
        }  
    } finally {  
        lock(LOCK) {  
            //Sanity check for null as this is a public method  
            if (sWakeLock != null) sWakeLock.Release();  
        }  
    }  
}  
  
private void HandleRegistration(Intent intent) {  
    string registrationId = intent.GetStringExtra("registration_id");  
    string error = intent.GetStringExtra("error");  
    string unregistration = intent.GetStringExtra("unregistered");  
  
    if (string.IsNullOrEmpty(error)) {  
        var response = await SnsClient.CreatePlatformEndpointAsync(new  
CreatePlatformEndpointRequest {  
            Token = registrationId,  
            PlatformApplicationArn = "YourPlatformArn" /* insert your platform application ARN        });  
    }  
}  
  
private void HandleMessage(Intent intent) {  
    string message = string.Empty;  
    Bundle extras = intent.Extras;  
    if (!string.IsNullOrEmpty(extras.GetString("message"))) {  
        message = extras.GetString("message");  
    } else {  
        message = extras.GetString("default");  
    }  
  
    Log.Info("Messages", "message received = " + message);  
    ShowNotification(this, "SNS Push", message);  
    //show the message  
  
}  
  
public void ShowNotification(string contentTitle,  
string contentText) {  
    // Intent  
    Notification.Builder builder = new Notification.Builder(this)  
        .SetContentTitle(contentTitle)  
        .SetContentText(contentText)  
        .SetDefaults(NotificationDefaults.Sound | NotificationDefaults.Vibrate)  
        .SetSmallIcon(Resource.Drawable.Icon)  
        .SetSound(RingtoneManager.GetDefaultUri(RingtoneType.Notification));  
  
    // Get the notification manager:  
    NotificationManager notificationManager =  
this.GetSystemService(Context.NotificationService) as NotificationManager;  
  
    notificationManager.Notify(1001, builder.Build());  
}  
}
```

Send a Message from the SNS Console to Your Endpoint

1. Go to the [SNS Console > Applications](#).
2. Select your platform application, select an endpoint, and click **Publish to endpoint**.
3. Type in a text message in the text box and click **Publish message** to publish a message.

Amazon Cognito Identity

What is Amazon Cognito Identity?

Amazon Cognito Identity enables you to create unique identities for your users and authenticate them with identity providers. With an identity, you can obtain temporary, limited-privilege AWS credentials to synchronize data with Amazon Cognito Sync, or directly access other AWS services. Amazon Cognito Identity supports public identity providers—Amazon, Facebook, and Google—as well as unauthenticated identities. It also supports developer authenticated identities, which let you register and authenticate users via your own backend authentication process.

For more information on Cognito Identity, see the [Amazon Cognito Developer Guide](#).

For information about Cognito Authentication Region availability, see [AWS Service Region Availability](#).

Using a Public Provider to Authenticate Users

Using Amazon Cognito Identity, you can create unique identities for your users and authenticate them for secure access to your AWS resources like Amazon S3 or Amazon DynamoDB. Amazon Cognito Identity supports public identity providers—Amazon, Facebook, Twitter/Digits, Google, or any OpenID Connect-compatible provider—as well as unauthenticated identities.

For information on using public identity providers like Amazon, Facebook, Twitter/Digits, or Google to authenticate users, see the [External Providers](#) in the Amazon Cognito Developer Guide.

Using Developer Authenticated Identities

Amazon Cognito supports developer authenticated identities, in addition to web identity federation through Facebook, Google, and Amazon. With developer authenticated identities, you can register and authenticate users via your own existing authentication process, while still using [Amazon Cognito Sync \(p. 28\)](#) to synchronize user data and access AWS resources. Using developer authenticated identities involves interaction between the end user device, your backend for authentication, and Amazon Cognito.

For information on developer authenticated identities, see the [Developer Authenticated Identities](#) in the Amazon Cognito Developer Guide.

Amazon Cognito Sync

What is Amazon Cognito Sync?

Cognito Sync is an AWS service and client library that enables cross-device syncing of user data (e.g. game scores, user preferences, game state). You can use the Cognito Sync API to synchronize user data across devices. To use Cognito Sync in your app, you must include the | in your project.

For instructions on how to integrate Amazon Cognito Sync in your application, see [Amazon Cognito Sync Developer Guide](#).

Amazon Mobile Analytics

[Amazon Mobile Analytics](#) is a service for collecting, visualizing, understanding and extracting app usage data at scale. Mobile Analytics easily captures both standard device data and custom events and automatically calculates reports on your behalf. In addition to the aggregated reports listed below, you can also setup your data to be exported to Redshift and S3 automatically for further analysis.

Using Amazon Mobile Analytics, you can track customer behaviors, aggregate metrics, generate data visualizations, and identify meaningful patterns.

Key Concepts

Report Types

Out of the box, Mobile Analytics provides the following reports in the Mobile Analytics Console:

- Daily Active Users (DAU), Monthly Active Users (MAU), and New Users
- Sticky Factor (DAU divided by MAU)
- Session Count and Average Sessions per Daily Active User
- Average Revenue per Daily Active User (ARPPDAU) and Average Revenue per Daily Paying Active User (ARPPDAU)
- Day 1, 3, and 7 Retention and Week 1, 2, and 3 Retention
- Custom Events

These reports are provided via six reporting tabs in the console:

- **Overview** – Track nine preselected reports in a simple-to-review dashboard to get a quick idea of engagement: MAU, DAU, New Users, Daily Sessions, Sticky Factor, 1-Day Retention, ARPPDAU, Daily Paying Users, ARPPDAU.
- **Active Users** – Track how many users engage with your app daily and monthly and monitor sticky factor to gauge engagement, appeal, and monetization.
- **Sessions** – Track how often your app is used on a given day and how often each user opens your app during a day.
- **Retention** – Track the rate at which customers come back to your app on a daily and weekly basis.
- **Revenue** – Track in-app revenue trends to identify areas for monetization improvement.
- **Custom events** – Track custom defined user actions specific to your app.

To learn more about Mobile Analytics reports and working in the Mobile Analytics console, see the [Mobile Analytics Console Reports Overview](#) in the Mobile Analytics Developer Guide.

Project Setup

Prerequisites

To use Mobile Analytics in your application, you'll need to add the SDK to your project. To do so, follow the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).

Configure Mobile Analytics Settings

Mobile Analytics defines some settings that can be configured in the `awsconfig.xml` file:

```
var config = new MobileAnalyticsManagerConfig();
config.AllowUseDataNetwork = true;
config.DBWarningThreshold = 0.9f;
config.MaxDBSize = 5242880;
config.MaxRequestSize = 102400;
config.SessionTimeout = 5;
```

- **SessionTimeout** - If the app stays in background for a time greater than the `SessionTimeout` then Mobile Analytics client terminates the current session and a new session is created when the app comes back to the foreground. We recommend using values ranging from 5 to 10. The default value is 5.
- **MaxDBSize** - The maximum size of the database (in bytes) used for local storage of events. If the database size exceeds this value, additional events will be ignored. We recommend using values ranging from 1MB to 10MB. The default value is 5242880 (5MB).
- **DBWarningThreshold** - The Warning threshold. Valid values range between 0 - 1. If the values exceed the threshold warning logs will be generated. The default value is 0.9.
- **MaxRequestSize** - The maximum size of an HTTP request made to the Mobile Analytics service. The value is specified in bytes and can range between 1-512KB. The default value is 102400 (100KB). Do not use values larger than 512KB, this may cause the service to reject the HTTP request.
- **AllowUseDataNetwork** - A value indicating whether service call is allowed over a cellular data network. Use this option with caution as this may increase customer's data usage.

The settings shown above are the default values for each configuration item.

Integrating Mobile Analytics with Your Application

The sections below explain how to integrate Mobile Analytics with your app.

Create an App in the Mobile Analytics Console

Go to the [Amazon Mobile Analytics console](#) and create an app. Note the `appId` value, as you'll need it later. When you are creating an app in the Mobile Analytics Console you will need to specify your identity pool ID. For instructions on creating an identity pool, see [Setting Up the AWS Mobile SDK for .NET and Xamarin](#) (p. 4).

To learn more about working in the Mobile Analytics Console, see the [Mobile Analytics Console Reports Overview](#) in the Mobile Analytics Developer Guide.

Create a MobileAnalyticsManager Client

To initialize your `MobileAnalyticsManager`, call `GetOrCreateInstance` on your `MobileAnalyticsManager`, passing in your AWS credentials, your region, your Mobile Analytics application ID, and your optional config object:

```
// Initialize the MobileAnalyticsManager
analyticsManager = MobileAnalyticsManager.GetOrCreateInstance(
```

```
cognitoCredentials,  
RegionEndpoint.USEast1,  
APP_ID,  
config  
);
```

The APP_ID is generated for you during the app creation wizard. Both of these values must match those in the Mobile Analytics Console. The APP_ID is used to group your data in the Mobile Analytics console. To find your app ID after creating the app in the Mobile Analytics console, browse to the Mobile Analytics Console, click the gear icon in the upper right-hand corner of the screen. This will display the App Management page which lists all registered apps and their app IDs.

Record Monetization Events

The AWS Mobile SDK for .NET and Xamarin provides the `MonetizationEvent` class, which enables you generate monetization events to track purchases made within mobile applications. The following code snippet shows how to create a monetization event:

```
// Create the monetization event object  
MonetizationEvent monetizationEvent = new MonetizationEvent();  
  
// Set the details of the monetization event  
monetizationEvent.Quantity = 3.0;  
monetizationEvent.ItemPrice = 1.99;  
monetizationEvent.ProductId = "ProductId123";  
monetizationEvent.ItemPriceFormatted = "$1.99";  
monetizationEvent.Store = "Your-App-Store";  
monetizationEvent.TransactionId = "TransactionId123";  
monetizationEvent.Currency = "USD";  
  
// Record the monetization event  
analyticsManager.RecordEvent(monetizationEvent);
```

Record Custom Events

Mobile Analytics allows you to define custom events. Custom events are defined entirely by you; they help you track user actions specific to your app or game. For more information about Custom events, see [Custom-Events](#).

For this example, we'll say that our app is a game, and that we want to record an event when a user completes a level. Create a "LevelComplete" event by creating a new `AmazonMobileAnalyticsEvent` instance:

```
CustomEvent customEvent = new CustomEvent("LevelComplete");  
  
// Add attributes  
customEvent.AddAttribute("LevelName", "Level1");  
customEvent.AddAttribute("CharacterClass", "Warrior");  
customEvent.AddAttribute("Successful", "True");  
  
// Add metrics  
customEvent.AddMetric("Score", 12345);  
customEvent.AddMetric("TimeInLevel", 64);  
  
// Record the event  
analyticsManager.RecordEvent(customEvent);
```

Recording Sessions

Xamarin iOS

When the application loses focus you can pause the session. For iOS apps, in the AppDelegate.cs file, override `DidEnterBackground` and `WillEnterForeground` to call `MobileAnalyticsManager.PauseSession` and `MobileAnalyticsManager.ResumeSession` as shown in the following snippet:

```
public override void DidEnterBackground(UIApplication application)
{
    // ...
    _manager.PauseSession();
    // ...
}

public override void WillEnterForeground(UIApplication application)
{
    // ...
    _manager.ResumeSession();
    // ...
}
```

Xamarin Android

For Android apps call `MobileAnalyticsManager.PauseSession` in the `OnPause()` method and `MobileAnalyticsManager.ResumeSession` in the `OnResume()` method as shown in the following code snippet:

```
protected override void OnResume()
{
    _manager.ResumeSession();
    base.OnResume();
}

protected override void OnPause()
{
    _manager.PauseSession();
    base.OnPause();
}
```

By default, if the user switches focus away from the app for less than 5 seconds, and switches back to the app the session will be resumed. If the user switches focus away from the app for 5 seconds or longer, a new session will be created. This setting is configurable in the `aws_mobile_analytics.json` configuration file by setting the "SESSION_DELTA" property to the number of seconds to wait before creating a new session.

Amazon Simple Storage Service (S3)

What is S3?

[Amazon Simple Storage Service \(Amazon S3\)](#), provides developers with secure, durable, highly-scalable object storage. Amazon S3 is easy to use, with a simple web services interface to store and retrieve any amount of data from anywhere on the web. With Amazon S3, you pay only for the storage you actually use. There is no minimum fee and no setup cost.

Amazon S3 provides cost-effective object storage for a wide variety of use cases including cloud applications, content distribution, backup and archiving, disaster recovery, and big data analytics.

For information about AWS S3 Region availability, see [AWS Service Region Availability](#).

Key Concepts

Bucket

Every object you store in Amazon S3 resides in a bucket. You can use buckets to group related objects in the same way that you use a directory to group files in a file system. Buckets have properties, such as access permissions and versioning status, and you can specify the region where you want them to reside.

To learn more about S3 Buckets, see [Working with Buckets](#) in the S3 Developer Guide.

Objects

Objects are the data that you store in Amazon S3. Every object resides within a bucket you create in specific AWS region.

Objects stored in a region never leave the region unless you explicitly transfer them to another region. For example, objects stored in the EU (Ireland) region never leave it. The objects stored in an Amazon S3 region physically remain in that region. Amazon S3 does not keep copies or move it to any other region. However, you can access the objects from anywhere, as long as you have necessary permissions.

Objects can be any file type: images, backup data, movies, etc. An object can be as large as 5 TB. You can have an unlimited number of objects in a bucket.

Before you can upload an object into Amazon S3, you must have write permissions to a bucket. For more information on setting bucket permissions, see [Editing Bucket Permissions](#) in the S3 Developer Guide.

To learn more about S3 objects, see [Working with Objects](#) in the S3 Developer Guide.

Object Metadata

Each object in Amazon S3 has a set of key-value pairs that represents its metadata. There are two types of metadata:

- **System metadata** – Sometimes processed by Amazon S3, e.g., Content-Type, and Content-Length.
- **User metadata** – Never processed by Amazon S3. User metadata is stored with the object and returned with it. The maximum size for user metadata is 2 KB, and both the keys and their values must conform to US-ASCII standards.

To learn more about S3 object metadata, see [Editing Object Metadata](#).

Project Setup

Prerequisites

To use Amazon S3 in your application, you'll need to add the SDK to your project. To do so, follow the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).

Create an S3 Bucket

Amazon S3 stores your application's resources in Amazon S3 buckets - cloud storage containers that live in a specific [region](#). Each Amazon S3 bucket must have a globally unique name. You can use the [Amazon S3 Console](#) to create a bucket.

1. Sign in to the [Amazon S3 console](#) and click **Create Bucket**.
2. Enter a bucket name, select a region, and click **Create**.

Set Permissions for S3

The default IAM role policy grants your application access to Amazon Mobile Analytics and Amazon Cognito Sync. In order for your Cognito identity pool to access Amazon S3, you must modify the identity pool's roles.

1. Go to the [Identity and Access Management Console](#) and click **Roles** in the left-hand pane.
2. Type your identity pool name into the search box. Two roles will be listed: one for unauthenticated users and one for authenticated users.
3. Click the role for unauthenticated users (it will have unauth appended to your identity pool name).
4. Click **Create Role Policy**, select **Policy Generator**, and then click **Select**.
5. On the **Edit Permissions** page, enter the settings shown in the following image, replacing the Amazon Resource Name (ARN) with your own. The ARN of an S3 bucket looks like `arn:aws:s3:::examplebucket/*` and is composed of the region in which the bucket is located and the name of the bucket. The settings shown below will give your identity pool full to access to all actions for the specified bucket.

Edit Permissions

The policy generator enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see [Overview of Policies](#) in Using AWS Identity and Access Management.

The screenshot shows the 'Edit Permissions' page in the AWS IAM console. At the top, there are radio buttons for 'Effect' with 'Allow' selected and 'Deny' unselected. Below this is a dropdown menu for 'AWS Service' set to 'Amazon S3'. Under 'Actions', a box shows 'All Actions Selected'. The 'Amazon Resource Name (ARN)' field contains 'arn:aws:s3:::examplebucket/*'. Below the ARN field is a link 'Add Conditions (optional)'. At the bottom is a button labeled 'Add Statement'.

1. Click the **Add Statement** button and then click **Next Step**.

2. The Wizard will show you the configuration that you generated. Click **Apply Policy**.

For more information on granting access to S3, see [Granting Access to an Amazon S3 Bucket](#).

(optional) Configure the Signature Version for S3 Requests

Every interaction with Amazon S3 is either authenticated or anonymous. AWS uses the Signature Version 4 or Signature Version 2 algorithms to authenticate calls to the service.

All new AWS regions created after January 2014 only support Signature Version 4. However, many older regions still support Signature Version 4 and Signature Version 2 requests.

If your bucket is in one of the regions that does not support Signature Version 2 requests as listed on [this page](#), you must set the `AWSConfigsS3.UseSignatureVersion4` property to "true" like so:

```
AWSConfigsS3.UseSignatureVersion4 = true;
```

For more information on AWS Signature versions, see [Authenticating Requests \(AWS Signature Version 4\)](#).

Integrating S3 with Your Application

There are two ways to interact with S3 in your Xamarin application. The two methods are explored in-depth in the following topics:

Using the S3 Transfer Utility

The S3 Transfer Utility makes it easier to upload and download files to S3 from your Xamarin application.

Initialize the TransferUtility

Create an S3 client, passing it your AWS credentials object, and then pass the S3 client to the transfer utility, like so:

```
var s3Client = new AmazonS3Client(credentials,region);  
var transferUtility = new TransferUtility(s3Client);
```

(optional) Configure the TransferUtility

There are three optional properties that you can configure:

- **ConcurrentServiceRequests** - Determines how many active threads or the number of concurrent asynchronous web requests will be used to upload/download the file. The default value is 10.
- **MinSizeBeforePartUpload** - Gets or sets the minimum part size for upload parts in bytes. The default is 16 MB. Decreasing the minimum part size causes multipart uploads to be split into a larger number of smaller parts. Setting this value too low has a negative effect on transfer speeds, causing extra latency and network communication for each part.
- **NumberOfUploadThreads** - Gets or sets the number of executing threads. This property determines how many active threads will be used to upload the file. The default value is 10 threads.

To configure the S3 TransferUtility client, create a config object, set your properties, and pass the object to your TransferUtility constructor like so:

```
var config = new TransferUtilityConfig();

config.ConcurrentServiceRequests = 10;
config.MinSizeBeforePartUpload=16*1024*1024;
config.NumberOfUploadThreads=10;

var s3Client = new AmazonS3Client(credentials);
var utility = new TransferUtility(s3Client,config);
```

Download a File

To download a file from S3, call `Download` on the Transfer Utility object, passing the following parameters:

- `file` - String name of the file that you want to download
- `bucketName` - String name of the S3 bucket from which you want to download the file
- `key` - A string representing the name of the S3 object (a file in this case) to download

```
transferUtility.Download(
    Path.Combine(Environment.SpecialFolder.ApplicationData, "file"),
    "bucketName",
    "key"
);
```

Upload a File

To upload a file to S3, call `Upload` on the Transfer Utility object, passing the following parameters:

- `file` - String name of the file that you want to upload
- `bucketName` - String name of the S3 bucket to store the file

```
transferUtility.Upload(
    Path.Combine(Environment.SpecialFolder.ApplicationData, "file"),
    "bucketName"
);
```

The code above assumes that there is a file in the directory `Environment.SpecialFolder.ApplicationData`. Uploads automatically use S3's multi-part upload functionality on large files to enhance throughput.

Using the Service Level S3 APIs

In addition to using the S3 TransferUtility, you can also interact with S3 using the low-level S3 APIs.

Initialize the Amazon S3 Client

To use Amazon S3, we first need to create an `AmazonS3Client` instance which takes a reference to the `CognitoAWSCredentials` instance you created previously and your region:

```
AmazonS3Client S3Client = new AmazonS3Client (credentials,region);
```

Download a File

To download a file from S3:

```
// Create a GetObject request
GetObjectRequest request = new GetObjectRequest
{
    BucketName = "SampleBucket",
    Key = "Item1"
};

// Issue request and remember to dispose of the response
using (GetObjectResponse response = client.GetObject(request))
{
    using (StreamReader reader = new StreamReader(response.ResponseStream))
    {
        string contents = reader.ReadToEnd();
        Console.WriteLine("Object - " + response.Key);
        Console.WriteLine(" Version Id - " + response.VersionId);
        Console.WriteLine(" Contents - " + contents);
    }
}
```

Upload a File

To upload a file to S3:

```
// Create a client
AmazonS3Client client = new AmazonS3Client();

// Create a PutObject request
PutObjectRequest request = new PutObjectRequest
{
    BucketName = "SampleBucket",
    Key = "Item1",
    FilePath = "contents.txt"
};

// Put object
PutObjectResponse response = client.PutObject(request);
```

Delete an Item

To delete an item in S3:

```
// Create a client
AmazonS3Client client = new AmazonS3Client();

// Create a DeleteObject request
DeleteObjectRequest request = new DeleteObjectRequest
{
    BucketName = "SampleBucket",
    Key = "Item1"
};

// Issue request
```

```
client.DeleteObject(request);
```

Delete Multiple Items

To delete multiple objects from a bucket using a single HTTP request:

```
// Create a client
AmazonS3Client client = new AmazonS3Client();

// Create a DeleteObject request
DeleteObjectsRequest request = new DeleteObjectsRequest
{
    BucketName = "SampleBucket",
    Objects = new List<KeyVersion>
    {
        new KeyVersion() {Key = "Item1"},
        // Versioned item
        new KeyVersion() { Key = "Item2", VersionId =
"Rej8CiBxcZKVK81cLr39j27Y5FVXghDK", },
        // Item in subdirectory
        new KeyVersion() { Key = "Logs/error.txt"}
    }
};

try
{
    // Issue request
    DeleteObjectsResponse response = client.DeleteObjects(request);
}
catch (DeleteObjectsException doe)
{
    // Catch error and list error details
    DeleteObjectsResponse errorResponse = doe.Response;

    foreach (DeletedObject deletedObject in errorResponse.DeletedObjects)
    {
        Console.WriteLine("Deleted item " + deletedObject.Key);
    }
    foreach (DeleteError deleteError in errorResponse.DeleteErrors)
    {
        Console.WriteLine("Error deleting item " + deleteError.Key);
        Console.WriteLine(" Code - " + deleteError.Code);
        Console.WriteLine(" Message - " + deleteError.Message);
    }
}
```

You may specify up to 1000 keys.

List Buckets

To return a list of all buckets owned by the authenticated sender of the request:

```
// Create a client
AmazonS3Client client = new AmazonS3Client();

// Issue call
ListBucketsResponse response = client.ListBuckets();

// View response data
Console.WriteLine("Buckets owner - {0}", response.Owner.DisplayName);
foreach (S3Bucket bucket in response.Buckets)
```

```
{  
    Console.WriteLine("Bucket {0}, Created on {1}", bucket.BucketName,  
        bucket.CreationDate);  
}
```

List Objects

You can return some or all (up to 1000) of the objects stored in your S3 bucket. To do so, you must have read access to the bucket.

```
// Create a GetObject request  
GetObjectRequest request = new GetObjectRequest  
{  
    BucketName = "SampleBucket",  
    Key = "Item1"  
};  
  
// Issue request and remember to dispose of the response  
using (GetObjectResponse response = client.GetObject(request))  
{  
    using (StreamReader reader = new StreamReader(response.ResponseStream))  
    {  
        string contents = reader.ReadToEnd();  
        Console.WriteLine("Object - " + response.Key);  
        Console.WriteLine(" Version Id - " + response.VersionId);  
        Console.WriteLine(" Contents - " + contents);  
    }  
}
```

Get a Bucket's Region

To obtain the region that a bucket resides in:

```
// Create a client  
AmazonS3Client client = new AmazonS3Client();  
  
// Construct request  
GetBucketLocationRequest request = new GetBucketLocationRequest  
{  
    BucketName = "SampleBucket"  
};  
  
// Issue call  
GetBucketLocationResponse response = client.GetBucketLocation(request);  
  
// View response data  
Console.WriteLine("Bucket location - {0}", response.Location);
```

Get a Bucket's Policy

To get a bucket's policy:

```
// Create a client  
AmazonS3Client client = new AmazonS3Client();  
  
// Construct request  
GetBucketPolicyRequest getRequest = new GetBucketPolicyRequest  
{
```

```
        BucketName = "SampleBucket"  
    };  
    string policy = client.GetBucketPolicy(getRequest).Policy;  
  
    Console.WriteLine(policy);  
    Debug.Assert(policy.Contains("BasicPerms"));
```

Amazon DynamoDB

What is Amazon DynamoDB?

[Amazon DynamoDB](#) is a fast, highly scalable non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

Key Concepts

The DynamoDB data model concepts include tables, items, and attributes.

Tables

In Amazon DynamoDB, a database is a collection of tables. A table is a collection of items and each item is a collection of attributes.

In a relational database, a table has a predefined schema such as the table name, primary key, list of its column names and their data types. All records stored in the table must have the same set of columns. In contrast, DynamoDB only requires that a table has a primary key, but does not require you to define all of the attribute names and data types in advance.

To learn more about working with tables, see [Working with Tables in DynamoDB](#).

Items and Attributes

Individual items in a DynamoDB table can have any number of attributes, although there is a limit of 400 KB on the item size. An item size is the sum of lengths of its attribute names and values (binary and UTF-8 lengths).

Each attribute in an item is a name-value pair. An attribute can be single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed.

For example, consider storing a catalog of products in DynamoDB. You can create a table, ProductCatalog, with the Id attribute as its primary key. The primary key uniquely identifies each item, so that no two products in the table can have the same ID.

To learn more about working with items, see [Working with Items in DynamoDB](#).

Data Types

Amazon DynamoDB supports the following data types:

- **Scalar types** – Number, String, Binary, Boolean, and Null.
- **Multi-valued types** – String Set, Number Set, and Binary Set.
- **Document types** – List and Map.

For more information about Scalar Data Types, Multi-Valued Data Types, and Document Data Types, see [DynamoDB Data Types](#).

Primary Key

When you create a table, in addition to the table name, you must specify the primary key of the table. The primary key uniquely identifies each item in the table, so that no two items can have the same key. DynamoDB supports the following two types of primary keys:

- **Hash Key:** The primary key is made of one attribute, a hash attribute. DynamoDB builds an unordered hash index on this primary key attribute. Each item in the table is uniquely identified by its hash key value.
- **Hash and Range Key:** The primary key is made of two attributes. The first attribute is the hash attribute and the second one is the range attribute. DynamoDB builds an unordered hash index on the hash primary key attribute, and a sorted range index on the range primary key attribute. Each item in the table is uniquely identified by the combination of its hash and range key values. It is possible for two items to have the same hash key value, but those two items must have different range key values.

Secondary Indexes

When you create a table with a hash and range key, you can optionally define one or more secondary indexes on that table. A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key.

DynamoDB supports two kinds of secondary indexes: local secondary indexes and global secondary indexes.

- **Local secondary index:** An index that has the same hash key as the table, but a different range key.
- **Global secondary index:** An index with a hash and range key that can be different from those on the table.

You can define up to 5 global secondary indexes and 5 local secondary indexes per table. For more information, see [Improving Data Access with Secondary Indexes in DynamoDB](#) in the DynamoDB Developer Guide.

Query and Scan

In addition to using primary keys to access items, Amazon DynamoDB also provides two APIs for searching the data: Query and Scan. We recommend that you read [Guidelines for Query and Scan](#) in the DynamoDB Developer Guide to familiarize yourself with some best practices.

Query

A Query operation finds items in a table or a secondary index using only primary key attribute values. You must provide a hash key attribute name and a distinct value to search for. You can optionally provide a range key attribute name and value, and use a comparison operator to refine the search results.

For sample queries, see:

- [Using the Document Model \(p. 44\)](#)
- [Using the Object Persistence Model \(p. 47\)](#)
- [Using the DynamoDB Service Level APIs \(p. 50\)](#)

For more information on Query, see [Query](#) in the DynamoDB Developer Guide.

Scan

A Scan operation reads every item in a table or a secondary index. By default, a Scan operation returns all of the data attributes for every item in the table or index. You can use the `ProjectionExpression` parameter so that Scan only returns some of the attributes, rather than all of them.

For sample scans, see:

- [Using the Document Model \(p. 44\)](#)
- [Using the Object Persistence Model \(p. 47\)](#)
- [Using the DynamoDB Service Level APIs \(p. 50\)](#)

For more information on Scan, see [Scan](#) in the DynamoDB Developer Guide.

Project Setup

Prerequisites

To use DynamoDB in your application, you'll need to add the SDK to your project. To do so, follow the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).

Create a DynamoDB Table

To create a table, go to the [DynamoDB console](#) and follow these steps:

1. Click **Create Table**.
2. Enter the name of the table.
3. Select **Hash** as the primary key type.
4. Select a type and enter a value for the hash attribute name. Click **Continue**.
5. On the **Add Indexes** page, if you plan to use global secondary indexes, set **Index Type** to "Global Secondary Index" and under **Index Hash Key**, enter a value for the secondary index. This will allow you to query and scan using both the primary index and secondary index. Click **Add Index To Table**, and then click **Continue**. To skip using global secondary indexes, click **Continue**.
6. Set the read and write capacity to your desired levels. For more information on configuring capacity, see [Provisioned Throughput in Amazon DynamoDB](#). Click **Continue**.
7. On the next screen, enter a notification email to create throughput alarms, if desired. Click **Continue**.
8. On the summary page, click **Create**. DynamoDB will create your database.

Set Permissions for DynamoDB

To use DynamoDB in an application, you must set the correct permissions. The following IAM policy allows the user to delete, get, put, query, scan, and update items in a specific DynamoDB table, which is identified by [ARN](#):

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DeleteItem",
```

```
        "dynamodb:GetItem",  
        "dynamodb:PutItem",  
        "dynamodb:Query",  
        "dynamodb:Scan",  
        "dynamodb:UpdateItem"  
    ],  
    "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"  
  }  
]  
}
```

You can modify policies in the [IAM console](#). You should add or remove allowed actions based on the needs of your app.

To learn more about IAM policies, see [Using IAM](#).

To learn more about DynamoDB-specific policies, see [Using IAM to Control Access to DynamoDB Resources](#) in the DynamoDB Developer Guide.

Integrating DynamoDB with your Application

The AWS Mobile SDK for .NET and Xamarin provides a high-level library for working with DynamoDB. You can also make requests directly against the low-level DynamoDB API, but for most use cases the high-level library is recommended. The `AmazonDynamoDBClient` is an especially useful part of the high-level library. Using this class, you can perform various create, read, update, and delete (CRUD) operations and execute queries.

The AWS Mobile SDK for .NET and Xamarin allows you to make calls using APIs from the AWS SDK for .NET to work with DynamoDB. All of the APIs are available in the `AWSSDK.dll`. For information about downloading the AWS SDK for .NET, see [AWS SDK for .NET](#).

There are three ways you can interact with DynamoDB in your Xamarin application:

- **Document Model:** This API provides wrapper classes around the low-level DynamoDB API to further simplify your programming tasks. The `Table` and `Document` are the key wrapper classes. You can use the document model for the data operations such as create, retrieve, update and delete items. The API is available in the `Amazon.DynamoDB.DocumentModel` namespace.
- **Object Persistence Model:** The Object Persistence API enables you to map your client-side classes to the DynamoDB tables. Each object instance then maps to an item in the corresponding tables. The `DynamoDBContext` class in this API provides methods for you to save client-side objects to a table, retrieve items as objects and perform query and scan. You can use the Object Persistence model for the data operations such as create, retrieve, update and delete items. You must first create your tables using the Service Client API and then use the object persistence model to map your classes to the tables. The API is available in the `Amazon.DynamoDB.DataModel` namespace.
- **Service Client API:** This is the protocol-level API that maps closely to the DynamoDB API. You can use this low-level API for all table and item operations such as create, update, delete table and items. You can also query and scan your tables. This API is available in the `Amazon.DynamoDB` namespace.

These three models are explored in-depth in the following topics:

Using the Document Model

The Document Model provides wrapper classes around the low-level .NET API. The `Table` and `Document` are the key wrapper classes. You can use the Document Model to create, retrieve, update and delete

items. To create, update and delete tables, you must use the low-level API. For instructions on how to use the low-level API, see [Using the DynamoDB Service Level APIs \(p. 50\)](#). The low-level API is available in the `Amazon.DynamoDB.DocumentModel` namespace.

To learn more about the Document Model, see [.NET Document Model](#).

Create a DynamoDB Client

To create a DynamoDB client:

```
var client = new AmazonDynamoDBClient(credentials, region);  
DynamoDBContext context = new DynamoDBContext(client);
```

CRUD Operations

Save an Item

Create an item:

```
Table table = Table.LoadTable(client, "Books");  
id = Guid.NewGuid().ToString();  
var books = new Document();  
books["Id"] = id;  
books["Author"] = "Mark Twain";  
books["Title"] = "Adventures of Huckleberry Finn";  
books["ISBN"] = "112-111111";  
books["Price"] = "10";
```

Save an item to a DynamoDB table:

```
var book = await table.PutItemAsync(books);
```

Retrieve an Item

To retrieve an item:

```
public async Task GetItemAsync(AWSCredentials credentials, RegionEndpoint region)  
{  
    var client = new AmazonDynamoDBClient(credentials, region);  
    Table books = Table.LoadTable(client, "Books");  
    var book = await books.GetItemAsync(id);  
}
```

Update an Item

To update an item:

```
public async Task UpdateItemAttributesAsync(AWSCredentials credentials, RegionEndpoint region)  
{  
    var book = new Document();  
    book["Id"] = id;  
    book["PageCount"] = "200";  
    var client = new AmazonDynamoDBClient(credentials, region);
```

```
        Table books = Table.LoadTable(client, "Books");
        Document updatedBook = await books.UpdateItemAsync(book);
    }
```

To update an item conditionally:

```
public async Task UpdateItemConditionallyAsync(AWSCredentials credentials, RegionEndpoint
region) {
    var book = new Document();
    book["Id"] = id;
    book["Price"] = "30";

    // For conditional price update, creating a condition expression.
    Expression expr = new Expression();
    expr.ExpressionStatement = "Price = :val";
    expr.ExpressionAttributeValues[":val"] = 10.00;

    var client = new AmazonDynamoDBClient(credentials, region);
    Table books = Table.LoadTable(client, "Books");

    Document updatedBook = await books.UpdateItemAsync(book);
}
```

Delete an Item

To delete an item:

```
public async Task DeleteItemAsync(AWSCredentials credentials, RegionEndpoint region)
{
    var client = new AmazonDynamoDBClient(credentials, region);
    Table books = Table.LoadTable(client, "Books");
    await books.DeleteItemAsync(id);
}
```

Query and Scan

To query and retrieve all books whose author is "Mark Twain":

```
public async Task QueryAsync(AWSCredentials credentials, RegionEndpoint region) {
    var client = new AmazonDynamoDBClient(credentials, region);
    Table books = Table.LoadTable(client, "Books");
    var search = books.Query(new QueryOperationConfig() {
        IndexName = "Author-Title-index",
        Filter = new QueryFilter("Author", QueryOperator.Equal, "Mark Twain")
    });
    Console.WriteLine("ScanAsync: printing query response");
    var documents = await search.GetRemainingAsync();
    documents.ForEach((d) => {
        PrintDocument(d);
    });
}
```

The scan example code below returns all books in our table:

```
public async Task ScanAsync(AWSCredentials credentials, RegionEndpoint region) {
    var client = new AmazonDynamoDBClient(credentials, region);
    Table books = Table.LoadTable(client, "Books");
    var search = books.Scan(new ScanOperationConfig() {
        ConsistentRead = true
    });
}
```

```
});  
Console.WriteLine("ScanAsync: printing scan response");  
var documents = await search.GetRemainingAsync();  
documents.ForEach((d) => {  
    PrintDocument(d);  
});  
}
```

Using the Object Persistence Model

The AWS Mobile SDK for .NET and Xamarin provides an Object Persistence model that enables you to map your client-side classes to a DynamoDB table. Each object instance then maps to an item in the corresponding table. To save your client-side objects to a table, the Object Persistence model provides the `DynamoDBContext` class, an entry point to DynamoDB. This class provides you a connection to DynamoDB and enables you to access tables, perform various CRUD operations, and execute queries.

The Object Persistence model does not provide an API to create, update, or delete tables. It provides only data operations. To create, update and delete tables, you must use the low-level API. For instructions on how to use the low-level API, see [Using the DynamoDB Service Level APIs \(p. 50\)](#).

Overview

The Object Persistence model provides a set of attributes to map client-side classes to tables, and properties/fields to table attributes. The Object Persistence model supports both the explicit and default mapping between class properties and table attributes.

- **Explicit mapping:** To map a property to a primary key, you must use the `DynamoDBHashKey` and `DynamoDBRangeKey` Object Persistence model attributes. Additionally, for the non-primary key attributes, if a property name in your class and the corresponding table attribute to which you want to map it are not the same, then you must define the mapping by explicitly adding the `DynamoDBProperty` attribute.
- **Default mapping** - By default, the Object Persistence model maps the class properties to the attributes with the same name in the table.

You do not have to map every single class property. You identify these properties by adding the `DynamoDBIgnore` attribute. Saving and retrieving an instance of an object would omit any property marked with this attribute.

Supported Data Types

The Object Persistence model supports a set of primitive .NET data types, collections, and arbitrary data types. The model supports the following primitive data types.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`, `double`, `float`
- `Int16`, `Int32`, `Int64`
- `SByte`
- `string`
- `UInt16`, `UInt32`, `UInt64`

The Object Persistence model also supports the .NET collection types with the following limitations:

- Collection type must implement ICollection interface.
- Collection type must be composed of the supported primitive types. For example, ICollection<string>, ICollection<bool>.
- Collection type must provide a parameter-less constructor.

For more information on the Object Persistence model, see [.NET Object Persistence Model](#).

Create a DynamoDB Client

To create a DynamoDB client:

```
var client = new AmazonDynamoDBClient(credentials,region);
DynamoDBContext context = new DynamoDBContext(client);
```

CRUD Operations

Save an Object

Create an object:

```
[DynamoDBTable("Books")]
public class Book {
    [DynamoDBHashKey] // Hash key.
    public string Id {
        get;
        set;
    }

    [DynamoDBGlobalSecondaryIndexHashKey]
    public string Author {
        get;
        set;
    }

    [DynamoDBGlobalSecondaryIndexRangeKey]
    public string Title {
        get;
        set;
    }
    public string ISBN {
        get;
        set;
    }
    public int Price {
        get;
        set;
    }
    public string PageCount {
        get;
        set;
    }
}

Book myBook = new Book
{
    Id = id,
```

```
        Author = "Charles Dickens",  
        Title = "Oliver Twist",  
        ISBN = "111-1111111001",  
        Price = 10,  
        PageCount = 300  
    };
```

Save an object to a DynamoDB table:

```
context.Save(myBook);
```

Retrieve an Object

To retrieve an object:

```
Book retrievedBook = context.Load<Book>(1);
```

Update an Object

To update an object:

```
Book retrievedBook = context.Load<Book>(1);  
retrievedBook.ISBN = "111-1111111001";  
context.Save(retrievedBook);
```

Delete an Object

To delete an object:

```
Book retrievedBook = context.Load<Book>(1);  
context.Delete(retrievedBook);
```

Query and Scan

To query and retrieve all books whose author is "Charles Dickens":

```
public async Task QueryAsync(AWSCredentials credentials, RegionEndpoint region) {  
    var client = new AmazonDynamoDBClient(credentials, region);  
    DynamoDBContext context = new DynamoDBContext(client);  
  
    var search = context.FromQueryAsync < Book > (new  
        Amazon.DynamoDBv2.DocumentModel.QueryOperationConfig() {  
            IndexName = "Author-Title-index",  
            Filter = new Amazon.DynamoDBv2.DocumentModel.QueryFilter("Author",  
                Amazon.DynamoDBv2.DocumentModel.QueryOperator.Equal, "Charles Dickens")  
        });  
  
    Console.WriteLine("items retrieved");  
  
    var searchResponse = await search.GetRemainingAsync();  
    searchResponse.ForEach((s) => {  
        Console.WriteLine(s.ToString());  
    });  
}
```

The scan example code below returns all books in our table:

```
public async Task ScanAsync(AWSCredentials credentials, RegionEndpoint region) {
    var client = new AmazonDynamoDBClient(credentials, region);
    DynamoDBContext context = new DynamoDBContext(client);

    var search = context.FromScanAsync < Book > (new
    Amazon.DynamoDBv2.DocumentModel.ScanOperationConfig() {
        ConsistentRead = true
    });

    Console.WriteLine("items retrieved");

    var searchResponse = await search.GetRemainingAsync();
    searchResponse.ForEach((s) => {
        Console.WriteLine(s.ToString());
    });
}
```

Using the DynamoDB Service Level APIs

The Dynamo Service Level APIs allow you to create, update and delete tables. You can also perform typical create, read, update, and delete (CRUD) operations on items in a table using this API.

Create a DynamoDB Client

To create a DynamoDB client:

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials,region);
```

CRUD Operations

Save an Item

To save an item to a DynamoDB table:

```
// Create a client
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials,region);

// Define item attributes
Dictionary<string, AttributeValue> attributes = new Dictionary<string, AttributeValue>();

// Author is hash-key
attributes["Author"] = new AttributeValue { S = "Mark Twain" };
attributes["Title"] = new AttributeValue { S = "The Adventures of Tom Sawyer" };
attributes["PageCount"] = new AttributeValue { N = "275" };
attributes["Price"] = new AttributeValue{N = "10.00"};
attributes["Id"] = new AttributeValue{N="10"};
attributes["ISBN"] = new AttributeValue{S="111-1111111"};

// Create PutItem request
PutItemRequest request = new PutItemRequest
{
    TableName = "Books",
    Item = attributes
};

// Issue PutItem request
```



```
var response = await client.PutItemAsync(request);
```

Retrieve an Item

To retrieve an item:

```
// Create a client
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials,region);

Dictionary<string, AttributeValue> key = new Dictionary<string, AttributeValue>
{
    { "Id", new AttributeValue { N = "10" } }
};

// Create GetItem request
GetItemRequest request = new GetItemRequest
{
    TableName = "Books",
    Key = key,
};

// Issue request
var result = await client.GetItemAsync(request);

// View response
Console.WriteLine("Item:");
Dictionary<string, AttributeValue> item = result.Item;
foreach (var keyValuePair in item)
{
    Console.WriteLine("Author := {0}", item["Author"]);
    Console.WriteLine("Title := {0}", item["Title"]);
    Console.WriteLine("Price:= {0}", item["Price"]);
    Console.WriteLine("PageCount := {0}", item["PageCount"]);
}
```

Update an Item

To update an item:

```
// Create a client
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials,region);

Dictionary<string, AttributeValue> key = new Dictionary<string, AttributeValue>
{
    { "Id", new AttributeValue { N = "10" } }
};

// Define attribute updates
Dictionary<string, AttributeValueUpdate> updates = new Dictionary<string,
    AttributeValueUpdate>();
// Add a new string to the item's Genres SS attribute
updates["Genres"] = new AttributeValueUpdate()
{
    Action = AttributeAction.ADD,
    Value = new AttributeValue { SS = new List<string> { "Bildungsroman" } }
};

// Create UpdateItem request
UpdateItemRequest request = new UpdateItemRequest
{
    TableName = "Books",
    Key = key,
```

```
        AttributeUpdates = updates
    };

    // Issue request
    var response = await client.UpdateItemAsync(request);
```

Delete an Item

To delete an item:

```
// Create a client
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials, region);

Dictionary<string, AttributeValue> key = new Dictionary<string, AttributeValue>
{
    { "Id", new AttributeValue { N = "10" } }
};

// Create DeleteItem request
DeleteItemRequest request = new DeleteItemRequest
{
    TableName = "Books",
    Key = key
};

// Issue request
var response = await client.DeleteItemAsync(request);
```

Query and Scan

To query and retrieve all books whose author is "Mark Twain":

```
public void Query(AWSCredentials credentials, RegionEndpoint region) {
    using(var client = new AmazonDynamoDBClient(credentials, region)) {
        var queryResponse = await client.QueryAsync(new QueryRequest() {
            TableName = "Books",
            IndexName = "Author-Title-index",
            KeyConditionExpression = "Author = :v_Id",
            ExpressionAttributeValues = new Dictionary < string, AttributeValue > {
                {
                    ":v_Id", new AttributeValue {
                        S = "Mark Twain"
                    }
                }
            }
        });
        queryResponse.Items.ForEach((i) => {
            Console.WriteLine(i["Title"].S);
        });
    }
}
```

The scan example code below returns all books in our table:

```
public void Scan(AWSCredentials credentials, RegionEndpoint region) {
    using(var client = new AmazonDynamoDBClient(credentials, region)) {
        var queryResponse = client.Scan(new ScanRequest() {
            TableName = "Books"
        });
    }
}
```

```
        queryResponse.Items.ForEach((i) => {  
            Console.WriteLine(i["Title"].S);  
        });  
    }  
}
```

Amazon Simple Notification Service (SNS)

Using SNS and the AWS Mobile SDK for .NET and Xamarin, you can write applications that can receive mobile push notifications. For information about SNS, see [Amazon Simple Notification Service](#).

Key Concepts

Amazon SNS allows applications and end-users on different devices to receive notifications via Mobile Push notification (Apple, Google and Kindle Fire Devices), HTTP/HTTPS, Email/Email-JSON, SMS or Amazon Simple Queue Service (SQS) queues, or AWS Lambda functions. SNS lets you send individual messages or to fan-out messages to large numbers of recipients subscribed to a single topic.

Topics

A topic is an “access point” for allowing recipients to dynamically subscribe for identical copies of the same notification. One topic can support deliveries to multiple endpoint types -- for example, you can group together iOS, Android and SMS recipients.

Subscriptions

To receive messages published to a topic, you have to subscribe an endpoint to that topic. An endpoint is a mobile app, web server, email address, or an Amazon SQS queue that can receive notification messages from Amazon SNS. Once you subscribe an endpoint to a topic and the subscription is confirmed, the endpoint will receive all messages published to that topic.

Publishing

When you publish to a topic, SNS delivers appropriately formatted copies of your message to each subscriber of that topic. For Mobile Push Notifications, you can publish directly to the endpoint or subscribe the endpoint to a topic.

Project Setup

Prerequisites

To use SNS in your application, you'll need to add the SDK to your project. To do so, follow the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin](#) (p. 4).

Set Permissions for SNS

For information on setting permissions for SNS, see [Managing Access to Your Amazon SNS Topics](#).

Add NuGet Package for SNS to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Amazon Simple Notification Service NuGet package to your project.

Integrating SNS with Your Application

There are many ways to interact with SNS in your Xamarin application:

Send Push Notifications (Xamarin Android)

This document explains how to send push notifications to a Xamarin Android application using Amazon Simple Notification Service (SNS) and the AWS Mobile SDK for .NET and Xamarin.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Set Permissions for SNS

Follow Step 2 in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to attach the policy mentioned below to your application's roles. This will give your application the proper permissions to access SNS:

1. Go to the [IAM Console](#) and select the IAM role that you want to configure.
2. Click **Attach Policy**, select the AmazonSNSFullAccess policy and click **Attach Policy**.

Warning

Using AmazonSNSFullAccess is not recommended in a production environment. We use it here to allow you to get up and running quickly. For more information about specifying permissions for an IAM role, see [Overview of IAM Role Permissions](#).

Enable Push Notifications on Google Cloud

First, add a new Google API project:

1. Go to the [Google Developers Console](#).
2. Click **Create Project**.
3. In the **New Project** box, enter a project name, take note of the project ID (you will need it later) and click **Create**.

Next, enable the Google Cloud Messaging (GCM) service for your project:

1. In the [Google Developers Console](#), your new project should already be selected. If not, select it in the drop-down at the top of the page.
2. Select **APIs & auth** from the side bar on the left-hand side of the page.

3. In the search box, type "Google Cloud Messaging for Android" and click the **Google Cloud Messaging for Android** link.
4. Click **Enable API**.

Finally, obtain an API Key:

1. In the Google Developers Console, select **APIs & auth > Credentials**.
2. Under **Public API access**, click **Create new key**.
3. In the **Create a new key** dialog, click **Server key**.
4. In the resulting dialog, click **Create** and copy the API key displayed. You will use this API key to perform authentication later on.

Use Project ID to Create a Platform ARN in SNS Console

1. Go to the [SNS Console](#).
2. Click **Applications** on the left-hand side of the screen.
3. Click **Create platform application** to create a new SNS platform application.
4. Enter an **Application Name**.
5. Select **Google Cloud Messaging (GCM)** for **Push notification platform**.
6. Paste the API key into the text box labeled **API key**.
7. Click **Create platform application**.
8. Select the Platform Application you just created and copy the Application ARN.

Add NuGet Package for SNS to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Amazon Simple Notification Service NuGet package to your project.

Create an SNS client

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

Register Your Application for Remote Notifications

In order to register for remote notifications on Android, you will need to create a BroadcastReceiver which can receive Google Cloud messages. Change the package name below where prompted to do so:

```
[BroadcastReceiver(Permission = "com.google.android.c2dm.permission.SEND")]
[IntentFilter(new string[] {
    "com.google.android.c2dm.intent.RECEIVE"
}, Categories = new string[] {
    "com.amazonaws.sns" /* change to match your package */
})]
[IntentFilter(new string[] {
    "com.google.android.c2dm.intent.REGISTRATION"
}, Categories = new string[] {
    "com.amazonaws.sns" /* change to match your package */
})]
[IntentFilter(new string[] {
    "com.google.android.gcm.intent.RETRY"
}, Categories = new string[] {
```

```
        "com.amazonaws.sns" /* change to match your package */
    }]]
    public class GCMBroadcastReceiver: BroadcastReceiver {
        const string TAG = "PushHandlerBroadcastReceiver";
        public override void OnReceive(Context context, Intent intent) {
            GCMIntentService.RunIntentInService(context, intent);
            SetResult(Result.Ok, null, null);
        }
    }

    [BroadcastReceiver]
    [IntentFilter(new[] {
        Android.Content.Intent.ActionBootCompleted
    })]
    public class GCMBootReceiver: BroadcastReceiver {
        public override void OnReceive(Context context, Intent intent) {
            GCMIntentService.RunIntentInService(context, intent);
            SetResult(Result.Ok, null, null);
        }
    }
}
```

Below is the service that receives the push notification from the BroadcastReceiver and displays the notification on the device's notification bar:

```
[Service]
public class GCMIntentService: IntentService {
    static PowerManager.WakeLock sWakeLock;
    static object LOCK = new object();

    public static void RunIntentInService(Context context, Intent intent) {
        lock(LOCK) {
            if (sWakeLock == null) {
                // This is called from BroadcastReceiver, there is no init.
                var pm = PowerManager.FromContext(context);
                sWakeLock = pm.NewWakeLock(
                    WakeLockFlags.Partial, "My WakeLock Tag");
            }
        }

        sWakeLock.Acquire();
        intent.SetClass(context, typeof(GCMIntentService));
        context.StartService(intent);
    }

    protected override void OnHandleIntent(Intent intent) {
        try {
            Context context = this.ApplicationContext;
            string action = intent.Action;

            if (action.Equals("com.google.android.c2dm.intent.REGISTRATION")) {
                HandleRegistration(intent);
            } else if (action.Equals("com.google.android.c2dm.intent.RECEIVE")) {
                HandleMessage(intent);
            }
        } finally {
            lock(LOCK) {
                //Sanity check for null as this is a public method
                if (sWakeLock != null) sWakeLock.Release();
            }
        }
    }

    private void HandleRegistration(Intent intent) {
        string registrationId = intent.GetStringExtra("registration_id");
    }
}
```

```
string error = intent.GetStringExtra("error");
string unregistration = intent.GetStringExtra("unregistered");

if (string.IsNullOrEmpty(error)) {
    var response = await SnsClient.CreatePlatformEndpointAsync(new
CreatePlatformEndpointRequest {
    Token = registrationId,
    PlatformApplicationArn = "YourPlatformArn" /* insert your platform application ARN
here */
    });
}

private void HandleMessage(Intent intent) {
    string message = string.Empty;
    Bundle extras = intent.Extras;
    if (!string.IsNullOrEmpty(extras.GetString("message"))) {
        message = extras.GetString("message");
    } else {
        message = extras.GetString("default");
    }

    Log.Info("Messages", "message received = " + message);
    ShowNotification(this, "SNS Push", message);
    //show the message
}

public void ShowNotification(string contentTitle,
string contentText) {
    // Intent
    Notification.Builder builder = new Notification.Builder(this)
        .SetContentTitle(contentTitle)
        .SetContentText(contentText)
        .SetDefaults(NotificationDefaults.Sound | NotificationDefaults.Vibrate)
        .SetSmallIcon(Resource.Drawable.Icon)
        .SetSound(RingtoneManager.GetDefaultUri(RingtoneType.Notification));

    // Get the notification manager:
    NotificationManager notificationManager =
this.GetService<Context, NotificationService>() as NotificationManager;

    notificationManager.Notify(1001, builder.Build());
}
```

Send a Message from the SNS Console to Your Endpoint

1. Go to the [SNS Console > Applications](#).
2. Select your platform application, select an endpoint, and click **Publish to endpoint**.
3. Type in a text message in the text box and click **Publish message** to publish a message.

Send Push Notifications (Xamarin iOS)

This document explains how to send push notifications to a Xamarin iOS application using Amazon Simple Notification Service (SNS) and the AWS Mobile SDK for .NET and Xamarin.

Project Setup

Prerequisites

You must complete all of the instructions on the [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) before beginning this tutorial.

Set Permissions for SNS

Follow Step 2 in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to attach the policy mentioned below to your application's roles. This will give your application the proper permissions to access SNS:

1. Go to the [IAM Console](#) and select the IAM role that you want to configure.
2. Click **Attach Policy**, select the AmazonSNSFullAccess policy and click **Attach Policy**.

Warning

Using AmazonSNSFullAccess is not recommended in a production environment. We use it here to allow you to get up and running quickly. For more information about specifying permissions for an IAM role, see [Overview of IAM Role Permissions](#).

Obtain Membership in the Apple iOS Developer Program

You will need to run your app on a physical device to receive push notifications. To run your app on a device, you must have a membership in the [Apple iOS Developer Program Membership](#). Once you have a membership, you can use Xcode to generate a signing identity. For more information, see Apple's [App Distribution Quick Start](#) documentation.

Create an iOS Certificate

First, you need to create an iOS Certificate. Then, you need to create a provisioning profile configured for push notifications. To do so:

1. Go to the [Apple Developer Member Center](#), click **Certificates, Identifiers & Profiles**.
2. Click **Identifiers** under **iOS Apps**, click the plus button in the upper right-hand corner of the web page to add a new iOS App ID, and enter an App ID description.
3. Scroll down to the **Add ID Suffix** section and select **Explicit App ID** and enter your bundle identifier.
4. Scroll down to the **App Services** section and select **Push Notifications**.
5. Click **Continue**.
6. Click **Submit**.
7. Click **Done**.
8. Select the App ID you just created and then click **Edit**.
9. Scroll down to the **Push Notifications** section. Click **Create Certificate** under **Development SSL Certificate**.
- 10 Follow the instructions to create a Certificate Signing Request (CSR), upload the request, and download an SSL certificate that will be used to communicate with Apple Notification Service (APNS).
- 11 Return to the **Certificates, Identifiers & Profiles** page. Click **All** under **Provisioning Profiles**.
- 12 Click the plus button in the upper right-hand corner to add a new provisioning profile.
- 13 Select **iOS App Development**, and then click **Continue**.

- 14 Select your App ID, and then click **Continue**.
- 15 Select your developer certificate, and then click **Continue**.
- 16 Select your device, and then click **Continue**.
- 17 Enter a profile name, and then click **Generate**.
- 18 Download and double-click the provision file to install the provisioning profile.

For more information on provisioning a profile configured for push notifications, see Apple's [Configuring Push Notifications](#) documentation.

Use Certificate to Create Platform ARN in SNS Console

1. Run the KeyChain access app, select **My Certificates** on the lower left-hand side of the screen, and then right-click the SSL certificate you generated to connect to APNS and select **Export**. You will be prompted to specify a name for the file and a password to protect the certificate. The certificate will be saved in a P12 file.
2. Go to the [SNS Console](#) and click **Applications** on the left-hand side of the screen.
3. Click **Create platform application** to create a new SNS platform application.
4. Enter an **Application Name**.
5. Select **Apple Development** for **Push notification platform**.
6. Click **Choose File** and select the P12 file you created when you exported your SSL certificate.
7. Enter the password you specified when you exported the SSL certificate and click **Load Credentials From File**.
8. Click **Create platform application**.
9. Select the Platform Application you just created and copy the Application ARN. You will need this in the upcoming steps.

Add NuGet Package for SNS to Your Project

Follow Step 4 of the instructions in [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#) to add the Amazon Simple Notification Service NuGet package to your project.

Create an SNS Client

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

Register Your Application for Remote Notifications

To register an application, call `RegisterForRemoteNotifications` on your `UIApplication` object, as shown below. Place the following code in `AppDelegate.cs`, inserting your platform application ARN where prompted below:

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options) {  
    // do something  
    var pushSettings = UIUserNotificationSettings.GetSettingsForTypes (  
        UIUserNotificationType.Alert |  
        UIUserNotificationType.Badge |  
        UIUserNotificationType.Sound,  
        null  
    );  
    RegisterForRemoteNotificationsWithApplicationIdentifier(PlatformApplicationARN);  
}
```

```
);  
app.RegisterUserNotifications(pushSettings);  
app.RegisterForRemoteNotifications();  
// do something  
    return true;  
}  
  
public override void RegisteredForRemoteNotifications(UIApplication application, NSData  
token) {  
    var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ", "");  
    if (!string.IsNullOrEmpty(deviceToken)) {  
        //register with SNS to create an endpoint ARN  
        var response = await SnsClient.CreatePlatformEndpointAsync(  
            new CreatePlatformEndpointRequest {  
                Token = deviceToken,  
                PlatformApplicationArn = "YourPlatformArn" /* insert your platform application ARN  
here */  
            });  
    }  
}
```

Send a Message from the SNS Console to Your Endpoint

1. Go to the [SNS Console > Applications](#).
2. Select your platform application, select an endpoint, and click **Publish to endpoint**.
3. Type in a text message in the text box and click **Publish message** to publish a message.

Send and Receive SMS Notifications

You can use Amazon Simple Notification Service (Amazon SNS) to send and receive Short Message Service (SMS) notifications to SMS-enabled mobile phones and smart phones.

Note

SMS notifications are currently supported for phone numbers in the United States. SMS messages can be sent only from topics created in the US East (N. Virginia) region. However, you can publish messages to topics that you create in the US East (N. Virginia) region from any other region.

Create a Topic

To create a topic:

1. In the Amazon SNS console, click **Create new topic**. The Create new topic dialog box appears.
2. In the Topic name box, type a topic name.
3. In the Display name box, type a display name. The topic must have a display name assigned to it because the first ten (10) characters of the display name are used as the initial part of the text message prefix. The display name you enter will appear in the confirmation message that SNS sends to the user (the display name below is "AMZN SMS").

Would you like to receive messages from AMZN SMS? Reply YES AMZN SMS to receive messages. Reply HELP or STOP. Msg&data rates may apply.

1. Click **Create topic**. The new topic appears in the Topics page.
2. Select the new topic and then click the topic ARN. The Topic Details page appears.
3. Copy the topic ARN, as you will need it when you subscribe to a topic in the next step.

```
arn:aws:sns:us-west-2:111122223333:MyTopic
```

Subscribe to a Topic Using the SMS Protocol

Create an SNS client, passing your credentials object and the region of your identity pool:

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

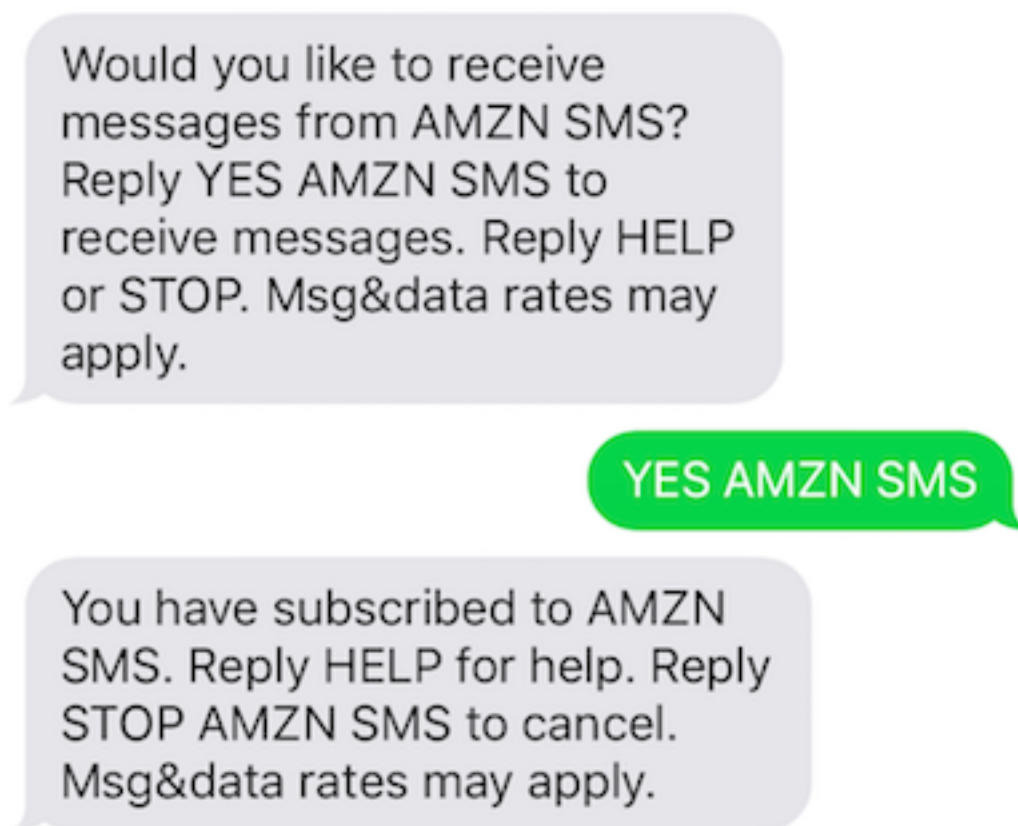
To subscribe to a topic, invoke `SubscribeAsync` and pass it the ARN of the topic that you want to subscribe to, the protocol ("sms"), and the phone number:

```
var response = await snsClient.SubscribeAsync(topicArn, "sms", "1234567890");
```

You will receive a subscribe arn in the subscribe response object. Your subscribe arn looks like this:

```
arn:aws:sns:us-west-2:123456789012:MyTopic:6b0e71bd-7e97-4d97-80ce-4a0994e55286
```

When a device subscribes to a topic, SNS will send a confirmation message to the device, and the user will have to confirm that they want to receive notifications, as shown below:

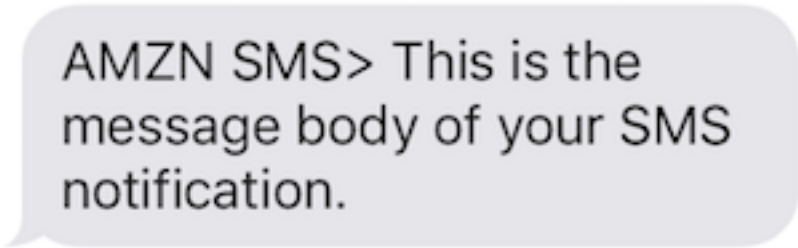


After the user subscribes to the topic, they will receive SMS messages when you publish them to that topic.

Publish a Message

To publish a message to a topic:

1. Sign in to the AWS Management Console and open the [Amazon SNS console](#).
2. In the left navigation pane, click **Topics** and then select the topic you want to publish to.
3. Click **Publish to topic**.
4. In the Subject box, type a subject.
5. In the Message box, type a message. Amazon SNS sends text that you enter in the Message box to SMS subscribers unless you also enter text into the Subject box. Because Amazon SNS includes a display name prefix with all SMS messages that you send, the sum of the display name prefix and the message payload cannot exceed 140 ASCII characters or 70 Unicode characters. Amazon SNS truncates messages that exceed these limits.
6. Click **Publish message**. Amazon SNS displays a confirmation dialog box. The SMS message appears on your SMS-enabled device, as shown below.



AMZN SMS> This is the message body of your SMS notification.

Send Messages to HTTP/HTTPS Endpoints

You can use Amazon SNS to send notification messages to one or more HTTP or HTTPS endpoints. The process is as follows:

1. Configure your endpoint to receive Amazon SNS messages.
2. Subscribe an HTTP/HTTPS endpoint to a topic.
3. Confirm your subscription.
4. Publish a notification to the topic. Amazon SNS then sends an HTTP POST request delivering the contents of the notification to the subscribed endpoint.

Configure Your HTTP/HTTPS Endpoint to Receive Amazon SNS Messages

Follow the instructions in Step 1 of [Sending Amazon SNS Messages to HTTP/HTTPS Endpoints](#) to configure your endpoint.

Subscribe Your HTTP/HTTPS endpoint to Your Amazon SNS Topic

Create an SNS client, passing your credentials object and the region of your identity pool:

```
var snsClient = new AmazonSimpleNotificationServiceClient(credentials, region);
```

To send messages to an HTTP or HTTPS endpoint through a topic, you must subscribe the endpoint to the Amazon SNS topic. You specify the endpoint using its URL:

```
var response = await snsClient.SubscribeAsync(  
    "topicArn",  
    "http", /* "http" or "https" */  
    "endpointUrl" /* endpoint url beginning with http or https */  
);
```

Confirm Your Subscription

After you subscribe to an endpoint, Amazon SNS will send a subscription confirmation message to the endpoint. The code at the endpoint must retrieve the `SubscribeURL` value from the subscription confirmation message and either visit the location specified by the `SubscribeURL` itself or make it available to you so that you can manually visit the `SubscribeURL` (for example, if using a web browser).

Amazon SNS will not send messages to the endpoint until the subscription is confirmed. When you visit the `SubscribeURL`, the response will contain an XML document containing an element `SubscriptionArn` that specifies the ARN for the subscription.

Send Messages to the HTTP/HTTPS Endpoint

You can send a message to a topic's subscriptions by publishing to the topic. Invoke `PublishAsync` and pass it the topic ARN and your message.

```
var response = await snsClient.PublishAsync(topicArn, "This is your message");
```

Troubleshooting SNS

Using Delivery Status in the Amazon SNS Console

The Amazon SNS console contains a Delivery Status feature that allows you to collect feedback on successful and unsuccessful delivery attempts of your messages to mobile push notification platforms (Apple (APNS), Google (GCM), Amazon (ADM), Windows (WNS and MPNS) and Baidu.

It also provides other important information such as dwell times in Amazon SNS. This information is captured in an Amazon CloudWatch Log group that is created automatically by Amazon SNS when this feature is enabled via the Amazon SNS console or via the Amazon SNS APIs.

For instructions on using the Delivery Status feature, see [Using the Delivery Status feature of Amazon SNS](#) on the AWS Mobile Blog.

Best Practices for Using the AWS Mobile SDK for .NET and Xamarin

There are only a few fundamentals and best practices that are helpful to know when using the AWS Mobile SDK for .NET and Xamarin.

- Use Amazon Cognito to obtain AWS credentials instead of hardcoding your credentials in your application. If you hard code your credentials in your application, you may end up exposing them to the public, allowing others to make calls to AWS using your credentials. For instructions on how to use Amazon Cognito to obtain AWS credentials, see [Setting Up the AWS Mobile SDK for .NET and Xamarin \(p. 4\)](#).
- For best practices on using S3, see [this article on the AWS Blog](#).
- For best practices on using DynamoDB, see [DynamoDB Best Practices](#) in the DynamoDB Developer Guide.

We are always looking to help our customers be successful and welcome feedback, so please feel free to [post on the AWS forums](#) or [open an issue on Github](#).

Library of AWS Service Documentation

Every service in the AWS Mobile SDK for .NET and Xamarin has a separate developer guide and service API reference which provides additional information that you might find helpful.

Amazon Cognito Identity

- [Cognito Developer Guide](#)
- [Cognito Identity Service API Reference](#)

Amazon Cognito Sync

- [Cognito Developer Guide](#)
- [Cognito Sync Service API Reference](#)

Amazon Mobile Analytics

- [Mobile Analytics Developer Guide](#)
- [Mobile Analytics Service API Reference](#)

Amazon S3

- [S3 Developer Guide](#)
- [S3 Getting Started Guide](#)

- [S3 Service API Reference](#)

Amazon DynamoDB

- [DynamoDB Developer Guide](#)
- [DynamoDB Getting Started Guide](#)
- [DynamoDB Service API Reference](#)

Amazon Simply Notification Service (SNS)

- [SNS Developer Guide](#)
- [SNS Service API Reference](#)

Other Helpful Links

- [AWS Glossary of Terms](#)
- [About AWS Credentials](#)

Troubleshooting

This topic describes some ideas for troubleshooting problems you might encounter when using the AWS Mobile SDK for .NET and Xamarin.

Ensure IAM Role Has Required Permissions

When calling AWS services, your app should be using an identity from a Cognito identity pool. Each identity in the pool is associated with an IAM (Identity and Access Management) role.

A role has one or more policy files associated with it that specify what AWS resources the users assigned to the role have access to. By default, two roles are created per identity pool: one for authenticated users and one for unauthenticated users.

You will need to either modify the existing policy file or associate a new policy file with the permissions required by your app. If your app allows both authenticated and unauthenticated users, both roles must be granted permissions for accessing the AWS resources your app needs.

The following policy file shows how to grant access to an S3 bucket:

```
{
  "Statement": [
    {
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::MYBUCKETNAME/*",
      "Principal": "*"
    }
  ]
}
```

The following policy file shows how to grant access to a DynamoDB database:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
    }
  ]
}
```

For more information about specifying policies, see [IAM Policies](#).

Using a HTTP Proxy Debugger

If the AWS service your app is calling has an HTTP or HTTPS endpoint, you can use an HTTP/HTTPS proxy debugger to view the requests and responses to gain more insight into what is occurring. There are a number of HTTP proxy debuggers available such as:

- [Charles](#) - a web debugging proxy for Windows and OSX
- [Fiddler](#) - a web debugging proxy for Windows

Both Charles and Fiddler require some configuration to be able to view SSL encrypted traffic, please read the documentation for these tools for further information. If you are using a web debugging proxy that cannot be configured to display encrypted traffic, open the `aws_endpoints.json` file and set the HTTP tag for the AWS service you need to debug to true.

Document History

The following table describes the important changes to the documentation since the last release of the AWS Mobile SDK for .NET and Xamarin.

- **API Version:** 2015-08-27
- **Last Documentation Update:** 2015-08-27

Change	API version	Description	Release date
GA release	2015-08-27	GA release	2015-08-27
Beta release	2015-07-28	Beta release	rn2015-07-28