AWS SDK for C++ Developer Guide

August 02, 2018

Contents

AWS C++ Developer Guide

AWS C++ Developer Guide

Welcome to the *AWS C++ Developer Guide*.

The AWS SDK for C++ provides a modern C++ (version C++ 11 or later) interface for Amazon Web Services (AWS). It provides both high-level and low-level APIs for nearly all AWS features, minimizing dependencies and providing platform portability on Windows, macOS, Linux, and mobile.

Additional Documentation and Resources

In addition to this guide, the following are valuable online resources for AWS SDK for C++ developers:

- *AWS SDK for C++ Reference*
- *Video:* Introducing the AWS SDK for C++ from AWS re:invent 2015
- AWS C++ Developer Blog
- GitHub:
    - SDK source
    - SDK issues
- SDK License

Getting Started

The topics in this section will help you set up and use the AWS SDK for C++.

Setting Up the SDK

Prerequisites

To use the AWS SDK for C++, you need:

- Visual Studio 2013 or later

> **Note**
>
> Visual Studio 2013 doesn't provide default move constructors and operators. Later versions of Visual Studio provide a standards-compliant compiler.

- *or* GNU Compiler Collection (GCC) 4.9 or later
- *or* Clang 3.3 or later
- A minimum of 4 GB of RAM

> **Note**
>
> You need 4 GB of RAM to build some of the larger AWS clients. The SDK might fail to build on Amazon EC2 instance types *t2.micro*, *t2.small*, and other small instance types due to insufficient memory.

Additional Requirements for Linux Systems

To compile on Linux, you must have the header files (`-dev` packages) for *libcurl*, *libopenssl*, *libuuid*, and *zlib*. Typically, you'll find the packages in your system's package manager.

To install these packages on *Debian/Ubuntu-based systems*

```
sudo apt-get install libcurl4-openssl-dev libssl-dev uuid-dev zlib1g-dev
```

To install these packages on *Redhat/Fedora-based systems*

```
sudo dnf install libcurl-devel openssl-devel libuuid-devel
```

Getting the SDK Using NuGet with Visual C++

You can use NuGet to manage dependencies for AWS SDK for C++ projects that you develop with Microsoft Visual C++. To use this procedure, you must have NuGet installed on your system.

To use the SDK with NuGet

1. Open your project in Visual Studio.
2. In **Solution Explorer**, right-click your project name and choose **Manage NuGet Packages**.
3. Select the packages to use by searching for a particular service or library name. For example, you could use a search term such as `aws s3 native` or, because AWS SDK for C++ libraries are named consistently, use `AWSSDKCPP-service name` to add a library for a particular service to your project.
4. Choose **Install** to install the libraries and add them to your project.

When you build your project, the correct binaries are automatically included for each runtime/architecture configuration you use—you won't need to manage these dependencies yourself.

Building the SDK from Source

If you don't use Visual Studio (or don't want to use NuGet), you can build the SDK from source to set it up on your development system. This method also enables you to customize your SDK build—see *CMake Parameters* for the available options.

To build the SDK from source

1. Download or clone the SDK source from aws/aws-sdk-cpp on GitHub.
   - Direct download: aws/aws-sdk-cpp/archive/master.zip
   - Clone with Git

     HTTPS

     ```
     git clone https://github.com/aws/aws-sdk-cpp.git
     ```

     SSH

     ```
     git clone git@github.com:aws/aws-sdk-cpp.git
     ```
2. Install cmake (*v3.0+*) and the relevant build tools for your platform. Ensure these are available in your PATH. If you're unable to install **cmake**, you can use **make** or **msbuild**.
3. Create a directory in which to create your buildfiles, and generate the necessary buildfiles within it. This is the recommended approach, referred to as an *out-of-source build*.:

   ```
   mkdir sdk_build
   cd sdk_build
   cmake <path/to/sdk/source>
   ```

   Alternatively, create the build files directly in the SDK source directory.:

   ```
   cd <path/to/sdk/source>
   cmake .
   ```

If you don't have **cmake** installed, you can use these alternative commands to set up your build directory:

auto make

```
make
```

Visual Studio

```
msbuild ALL_BUILD.vcxproj
```

4. Build and install the SDK by typing one of the following in the same location where you generated your build files:

auto make

```
make
sudo make install
```

Visual Studio

```
msbuild INSTALL.vcxproj
```

> **Tip**
>
> Building the entire SDK can take awhile. To build only a particular client such as Amazon S3, you can use the **cmake** *BUILD_ONLY* parameter. For example:
>
> ```
> cmake -DBUILD_ONLY="s3"
> ```
>
> See *CMake Parameters* for more ways to modify the build output.

Building for Android

To build for Android, add `-DTARGET_ARCH=ANDROID` to your **cmake** command line. The AWS SDK for C++ includes a **cmake** toolchain file that should cover what's needed, assuming you've set the appropriate environment variables (ANDROID_NDK).

Android on Windows

Building for Android on Windows requires additional setup. In particular, you have to run **cmake** from a Visual Studio (2013 or later) developer command prompt. You'll also need the commands **git** and **patch** in your path. If you have git installed on a Windows system, you'll most likely find **patch** in a sibling directory (.../Git/usr/bin/). Once you've verified these requirements, your **cmake** command line will change slightly to use **nmake**.:

```
cmake -G "NMake Makefiles" `-DTARGET_ARCH=ANDROID` <other options> ..
```

nmake builds targets in a serially. To make things go more quickly, we recommend installing JOM as an alternative to **nmake**, and then changing the **cmake** invocation as follows.:

```
cmake -G "NMake Makefiles JOM" `-DTARGET_ARCH=ANDROID` <other options> ..
```

Creating Release Builds

To create a *release* build of the SDK:

auto make

```
cmake -DCMAKE_BUILD_TYPE=Release <path/to/sdk/source>
make
sudo make install
```

Visual Studio

```
cmake <path-to-root-of-this-source-code> -G "Visual Studio 12 Win64"
msbuild INSTALL.vcxproj /p:Configuration=Release
```

Running Integration Tests

Several directories are appended with `*integration-tests`. After you build your project, you can run these executables to ensure everything works correctly.

Providing AWS Credentials

To connect to any of the supported services with the AWS SDK for C++, you must provide AWS credentials. The AWS SDKs and CLIs use *provider chains* to look for AWS credentials in several different places, including system/user environment variables and local AWS configuration files.

You can set your credentials for use by the AWS SDK for C++ in various ways, but here are the recommended approaches:

- Set credentials in the AWS credentials profile file on your local system, located at:
  - `~/.aws/credentials` on Linux, macOS, or Unix
  - `C:\Users\USERNAME\.aws\credentials` on Windows

  This file should contain lines in the following format:

  ```
  [default]
  aws_access_key_id = your_access_key_id
  aws_secret_access_key = your_secret_access_key
  ```

  Substitute your own AWS credentials values for the values *your_access_key_id* and *your_secret_access_key*.
- Set the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables.

  To set these variables on Linux, macOS, or Unix, use **export**:

```
export AWS_ACCESS_KEY_ID=your_access_key_id
export AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

To set these variables on Windows, use **set**:

```
set AWS_ACCESS_KEY_ID=your_access_key_id
set AWS_SECRET_ACCESS_KEY=your_secret_access_key
```

- For an EC2 instance, specify an IAM role and then give your EC2 instance access to that role. See IAM Roles for Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances* for a detailed discussion about how this works.

After you set your AWS credentials using one of these methods, the AWS SDK for C++ loads them automatically by using the default credential provider chain.

You can also supply AWS credentials using your own methods by:

- Providing credentials to an AWS client class constructor.
- Using Amazon Cognito, an AWS identity management solution. You can use the `CognitoCachingCredentialsProviders` classes in the identity-management project. For more information, see the *Amazon Cognito Developer Guide*.

Using the SDK

To use the SDK, you should properly initialize it with `Aws::InitAPI` before creating service clients and using them. You should then shut down the SDK with `Aws::ShutdownAPI`.

Both of these functions take an instance of Aws::SDKOptions, which you can use to set additional run-time options for SDK calls.

Initializing and Shutting Down the SDK

A basic skeleton application looks like this:

```
#include <aws/core/Aws.h>
int main(int argc, char** argv)
{
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        // make your SDK calls here.
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

best practice

To properly shut down / clean up any service clients that you may initialize before `Aws::ShutdownAPI` is called, it's a *best practice* to make sure that all SDK calls are made within a pair of curly-braces as shown above, or within another function called between `Aws::InitAPI` and `Aws::ShutdownAPI`.

Setting SDK options

The Aws::SDKOptions struct shown in Initializing and Shutting Down the SDK provides a number of options you can set. You should send the same options object to both `Aws::InitAPI` and `Aws::ShutdownAPI`.

A few examples:

- Turn logging on using the default logger:

```
Aws::SDKOptions options
options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Info;
Aws::InitAPI(options);
{
    // make your SDK calls here.
}
Aws::ShutdownAPI(options);
```

- Install a custom memory manager:

```
MyMemoryManager memoryManager;
Aws::SDKOptions options
options.memoryManagementOptions.memoryManager = &memoryManager;
Aws::InitAPI(options);
{
    // make your SDK calls here.
}
Aws::ShutdownAPI(options);
```

- Override the default HTTP client factory:

```
Aws::SDKOptions options
options.httpOptions.httpClientFactory_create_fn = [](){
        return Aws::MakeShared<MyCustomHttpClientFactory>(
            "ALLOC_TAG", arg1);
    };
Aws::InitAPI(options);
{
    // make your SDK calls here.
}
Aws::ShutdownAPI(options);
```

> **Note**
>
> `httpOptions` takes a closure instead of a `std::shared_ptr`. The SDK does this for all of its factory functions because the memory manager will not yet be installed at the time you will need to allocate this memory. Pass a closure to the SDK, and it will be called when it is safe to do so. This simplest way to do this is with a Lambda expression.

More Information

For further examples of AWS SDK for C++ application code, view the topics in the *Working with AWS Services* section. Each example contains a link to the full source code on GitHub, which you can use as a starting point for your own applications.

Building Your Application with CMake

CMake is a build tool that can manage your application's dependencies and create native makefiles suitable for the platform you're building on. It's an easy way to create and build projects using the AWS SDK for C++.

Setting Up a CMake Project

To set up a CMake project for use with the AWS SDK for C++

1. Create a directory to hold your source files.:

   ```
   mkdir my_example_project
   ```

2. Open the directory and add a CMakeLists.txt file that specifies your project's name, executables, source files, and linked libraries. The following is a minimal example:

   ```
   # minimal CMakeLists.txt for the AWS SDK for C++
   cmake_minimum_required(VERSION 2.8)

   # "my-example" is just an example value.
   project(my-example)

   # Locate the AWS SDK for C++ package.
   # Requires that you build with:
   #   -Daws-sdk-cpp_DIR=/path/to/sdk_build
   # or export/set:
   #   CMAKE_PREFIX_PATH=/path/to/sdk_build
   find_package(aws-sdk-cpp)

   # Link to the SDK shared libraries.
   add_definitions(-DUSE_IMPORT_EXPORT)

   # The executable name and its sourcefiles
   ```

After **cmake** generates your build directory, you can use **make** (or **nmake** on Windows) to build your application.:

```
make
```

Configuring the SDK

This section presents information about how to configure the AWS SDK for C++.

CMake Parameters

Use the CMake parameters listed in this section to customize how your SDK builds.

You can set these options with CMake GUI tools or the command line by using *-D*. For example:

```
cmake -DENABLE_UNITY_BUILD=ON -DREGENERATE_CLIENTS=1
```

General CMake Variables and Options

The following are general **cmake** variables and options that affect your SDK build.

> **Note**
>
> To use the *ADD_CUSTOM_CLIENTS* or *REGENERATE_CLIENTS* variables, you must have Python 2.7, Java (JDK 1.8+), and Maven installed and in your PATH.

ADD_CUSTOM_CLIENTS

Builds any arbitrary clients based on the API definition. Place your definition in the code-generation/api-definitions folder, and then pass this argument to **cmake**. The **cmake** configure step generates your client and includes it as a subdirectory in your build. This is particularly useful to generate a C++ client for using one of your API Gateway services. For example:

```
-DADD_CUSTOM_CLIENTS="serviceName=myCustomService;version=2015-12-21;serviceName=someOtherS
```

BUILD_ONLY

Builds only the clients you want to use. If set to a high-level SDK such as `aws-cpp-sdk-transfer`, *BUILD_ONLY* resolves any low-level client dependencies. It also builds integration and unit tests related to the projects you select, if they exist. This is a list argument, with values separated by semicolon (`;`) characters. For example:

```
-DBUILD_ONLY="s3;cognito-identity"
```

> **Note**
>
> The core SDK module, `aws-sdk-cpp-core`, is *always* built, regardless of the value of the *BUILD_ONLY* parameter.

BUILD_SHARED_LIBS

A built-in CMake option, re-exposed here for visibility. If enabled, it builds shared libraries; otherwise, it builds only static libraries.

> **Note**
>
> To dynamically link to the SDK, you must define the `USE_IMPORT_EXPORT` symbol for all build targets using the SDK.

Values: *ON* | *OFF*
Default: *ON*

CPP_STANDARD

Specifies a custom C++ standard for use with C++ 14 and 17 code bases.

Values: *11* | *14* | *17*
Default: *11*

CUSTOM_MEMORY_MANAGEMENT

To use a custom memory manager, set the value to `1`. You can install a custom allocator so that all STL types use the custom allocation interface. If you set the value `0`, you still might want to use the STL template types to help with DLL safety on Windows.

If static linking is enabled, custom memory management defaults to *off* (`0`). If dynamic linking is enabled, custom memory management defaults to *on* (`1`) and avoids cross-DLL allocation and deallocation.

> **Note**
>
> To prevent linker mismatch errors, you must use the same value (`0` or `1`) throughout your build system.

To install your own memory manager to handle allocations made by the SDK, you must set `-DCUSTOM_MEMORY_MANAGEMENT` and define `AWS_CUSTOM_MEMORY_MANAGEMENT` for all build targets that depend on the SDK.

ENABLE_RTTI

Controls whether the SDK is built to enable run-time type information (RTTI).

Values: *ON* | *OFF*
Default: *ON*

ENABLE_TESTING

Controls whether unit and integration test projects are built during the SDK build.

Values: *ON* | *OFF*
Default: *ON*

ENABLE_UNITY_BUILD

If enabled, most SDK libraries are built as a single, generated .cpp file. This can significantly reduce static library size and speed up compilation time.

Values: *ON* | *OFF*
Default: *OFF*

FORCE_SHARED_CRT

If enabled, the SDK links to the C runtime *dynamically*; otherwise, it uses the *BUILD_SHARED_LIBS* setting (sometimes necessary for backward compatibility with earlier versions of the SDK).

Values: *ON* | *OFF*
Default: *ON*

G

Generates build artifacts, such as Visual Studio solutions and Xcode projects.

For example, on Windows:

```
-G "Visual Studio 12 Win64"
```

For more information, see the CMake documentation for your platform.

MINIMIZE_SIZE

A superset of ENABLE_UNITY_BUILD. If enabled, this option turns on *ENABLE_UNITY_BUILD* and additional binary size reduction settings.

Values: *ON* | *OFF*
Default: *OFF*

NO_ENCRYPTION

If enabled, prevents the default platform-specific cryptography implementation from being built into the library. Turn this *ON* to inject your own cryptography implementation.

Values: *ON* | *OFF*
Default: *OFF*

NO_HTTP_CLIENT

If enabled, prevents the default platform-specific HTTP client from being built into the library. Turn this *ON* to inject your own HTTP client implementation.

Values: *ON* | *OFF*
Default: *OFF*

REGENERATE_CLIENTS

This argument wipes out all generated code and generates the client directories from the code-generation/api-definitions folder. For example:

```
-DREGENERATE_CLIENTS=1
```

SIMPLE_INSTALL

If enabled, the install process does not insert platform-specific intermediate directories underneath bin/ and lib/. Turn *OFF* if you need to make multiplatform releases under a single install directory.

Values: *ON* | *OFF*
Default: *ON*

TARGET_ARCH

To cross-compile or build for a mobile platform, you must specify the target platform. By default, the build detects the host operating system and builds for the detected operating system.

> ## Note
>
> When *TARGET_ARCH* is *ANDROID*, additional options are available. See Android CMake Variables and Options.

Values: *WINDOWS* | *LINUX* | *APPLE* | *ANDROID*

Android CMake Variables and Options

Use the following variables when you are creating an Android build of the SDK (when TARGET_ARCH is set to *ANDROID*).

ANDROID_ABI

Controls which Application Binary Interface (ABI) to output code for.

> ## Note
>
> Not all valid Android ABI values are currently supported.

Values: *arm64* | *armeabi-v7a* | *x86_64* | *x86* | *mips64* | *mips*
Default: *armeabi-v7a*

ANDROID_NATIVE_API_LEVEL

Controls what API level the SDK builds against. If you set ANDROID_STL to *gnustl*, you can choose any API level. If you use *libc++*, you must use an API level of at least *21*.

Default: Varies by STL choice.

ANDROID_STL

Controls what flavor of the C++ standard library the SDK uses.

Important

Performance problems can occur within the SDK if the `gnustl` options are used; we strongly recommend using *libc++_shared* or *libc++_static*.

Values: *libc++_shared* | *libc++_static* | *gnustl_shared* | *gnustl_static*
Default: *libc++_shared*

ANDROID_TOOLCHAIN_NAME

Controls which compiler is used to build the SDK.

Note

With GCC being deprecated by the Android NDK, we recommend using the default value.

Default: *standalone-clang*

DISABLE_ANDROID_STANDALONE_BUILD

By default, Android builds use a standalone clang-based toolchain constructed via NDK scripts. To use your own toolchain, turn this option *ON*.

Values: *ON* | *OFF*
Default: *OFF*

NDK_DIR

Specifies an override path where the build system should find the Android NDK. By default, the build system checks environment variables (ANDROID_NDK) if this variable is not set.

AWS Client Configuration

You can use the client configuration to control most functionality in the AWS SDK for C++.

`ClientConfiguration` declaration:

```
struct AWS_CORE_API ClientConfiguration
{
    ClientConfiguration();

    Aws::String userAgent;
    Aws::Http::Scheme scheme;
    Aws::Region region;
    Aws::String authenticationRegion;
```

```
    unsigned maxConnections;
    long requestTimeoutMs;
    long connectTimeoutMs;
    std::shared_ptr<RetryStrategy> retryStrategy;
    Aws::String endpointOverride;
    Aws::String proxyHost;
    unsigned proxyPort;
    Aws::String proxyUserName;
    Aws::String proxyPassword;
    std::shared_ptr<Aws::Utils::Threading::Executor> executor;
    bool verifySSL;
    Aws::String caPath;
    std::shared_ptr<Aws::Utils::RateLimits::RateLimiterInterface> writeRateLimiter;
    std::shared_ptr<Aws::Utils::RateLimits::RateLimiterInterface> readRateLimiter;
};
```

Configuration Variables

userAgent

Built in the constructor and pulls information from your operating system. Do not alter the user agent.

scheme

The default value is HTTPS. You can set this value to HTTP if the information you are passing is not sensitive and the service to which you want to connect supports an HTTP endpoint. AWS Auth protects you from tampering.

region

Specifies where you want the client to communicate. Examples include *us-east-1* or *us-west-1*. You must ensure that the service you want to use has an endpoint in the region you configure.

authenticationRegion

Allows you to specify an arbitrary region to use for signing. If you don't set `authenticationRegion`, we fall back to `region`. If you do set `authenticationRegion`, you are also responsible for setting endpoint override to connect to the endpoint that corresponds with your custom region.

maxConnections

The maximum number of allowed connections to a single server for your HTTP communications. The default value is 25. You can set this value as high as you can support the bandwidth. We recommend a value around 25.

requestTimeoutMs and connectTimeoutMs

Values that determine the length of time, in milliseconds, to wait before timing out a request. You can increase this value if you need to transfer large files, such as in Amazon S3 or Amazon CloudFront.

retryStrategy

Defaults to exponential backoff. You can override this default by implementing a subclass of `RetryStrategy` and passing an instance.

endpointOverride

Do not alter the endpoint.

proxyHost, proxyPort, proxyUserName, and proxyPassword

These settings allow you to configure a proxy for all communication with AWS. Examples of when this functionality might be useful include debugging in conjunction with the Burp suite, or using a proxy to connect to the Internet.

executor

The default behavior is to create and detach a thread for each async call. You can change this behavior by implementing a subclass of `Executor` and passing an instance.

verifySSL

Specifies whether to enable SSL certificate verification. If necessary, you can disable SSL certificate verification by setting `verifySSL` to `false`.

caPath

Enables you to tell the HTTP client where to find your SSL certificate trust store (for example, a directory prepared with OpenSSL's `c_rehash` utility). You shouldn't need to do this unless you are using symlinks in your environment. This has no effect on Windows or OS X.

writeRateLimiter and readRateLimiter

Used to throttle the bandwidth used by the transport layer. The default for these limiters is open. You can use the default implementation with the rates you want, or you can create your own instance by implementing a subclass of `RateLimiterInterface`.

Overriding Your HTTP Client

The default HTTP client for Windows is WinHTTP. The default HTTP client for all other platforms is curl. If needed, you can create a custom `HttpClientFactory` to pass to any service client's constructor.

Controlling IOStreams Used by the HttpClient and the AWSClient

By default, all responses use an input stream backed by a `stringbuf`. If needed, you can override the default behavior. For example, if you are using an Amazon S3 `GetObject` and don't want to load the entire file into memory, you can use `IOStreamFactory` in `AmazonWebServiceRequest` to pass a lambda to create a file stream.

Example file stream request

```
GetObjectRequest getObjectRequest;
getObjectRequest.SetBucket(fullBucketName);
getObjectRequest.SetKey(keyName);
getObjectRequest.SetResponseStreamFactory([](){
    return Aws::New<Aws::FStream>(
        ALLOCATION_TAG, DOWNLOADED_FILENAME, std::ios_base::out); });

auto getObjectOutcome = s3Client->GetObject(getObjectRequest);
```

Programming with the SDK

This section provides information about general use of the AWS SDK for C++, beyond that covered in *Getting Started*.

For service-specific programming examples, see *Working with AWS Services*.

Using Service Clients

AWS service client classes provide you with an interface to the AWS service that the class represents. Service clients follow the namespace convention `Aws::Service::ServiceClient`.

For example, a client for AWS Identity and Access Management is constructed using the `Aws::IAM::IAMClient` class. For an Amazon Simple Storage Service client, use `Aws::S3::S3Client`.

When you use the client classes to instantiate a service client, you must supply AWS credentials. You can do this by using the default credential provider chain, by manually passing credentials to the client directly, or by using a custom credentials provider.

For more information about setting credentials, see *Providing AWS Credentials*.

Using the Default Credential Provider Chain

The following code shows how to create an Amazon DynamoDB client by using a specialized client configuration, default credential provider chain, and default HTTP client factory.

```
auto limiter = Aws::MakeShared<Aws::Utils::RateLimits::DefaultRateLimiter<>>(ALLOCATION_TAG

// Create a client
ClientConfiguration config;
config.scheme = Scheme::HTTPS;
config.connectTimeoutMs = 30000;
config.requestTimeoutMs = 30000;
config.readRateLimiter = m_limiter;
config.writeRateLimiter = m_limiter;

auto client = Aws::MakeShared<DynamoDBClient>(ALLOCATION_TAG, config);
```

Passing Credentials Manually

The following code shows how to use the client constructor that takes three arguments, and use the `Aws::Auth::AWSCredentials` class to pass your credentials manually to the constructor.

```
auto client = Aws::MakeShared<DynamoDBClient>(
    ALLOCATION_TAG, AWSCredentials("access_key_id", "secret_key"), config);
```

Using a Custom Credentials Provider

The following code shows how to pass credentials to the `Aws::MakeShared` function and create a client by using one of the credential providers in the `Aws::Auth` namespace.

```
auto client = Aws::MakeShared<DynamoDBClient>(
    ALLOCATION_TAG,
    Aws::MakeShared<CognitoCachingAnonymousCredentialsProvider>(
        ALLOCATION_TAG, "identityPoolId", "accountId"), config);
```

Utility Modules

The AWS SDK for C++ provides you with several utility modules to ease the complexity of developing AWS applications in C++.

HTTP Stack

Headers: `/aws/core/http/`

The HTTP client provides connection pooling, is thread-safe, and can be reused as you need. For more information, see *AWS Client Configuration*.

String Utils

Header: `/aws/core/utils/StringUtils.h`

This header file provides core string functions, such as `trim`, `lowercase`, and numeric conversions.

Hashing Utils

Header: `/aws/core/utils/HashingUtils.h`

This header file provides hashing functions such as `SHA256`, `MD5`, `Base64`, and `SHA256_HMAC`.

JSON Parser

Header: `/aws/core/utils/json/JsonSerializer.h`

This header file provides a fully functioning yet lightweight JSON parser (thin wrapper around *JsonCpp*).

XML Parser

Header: `/aws/core/utils/xml/XmlSerializer.h`

This header file provides a lightweight XML parser (thin wrapper around *tinyxml2*). RAII pattern has been added to the interface.

Memory Management

The AWS SDK for C++ provides a way to control memory allocation and deallocation in a library.

> ### Note
>
> Custom memory management is available only if you use a version of the library built using the defined compile-time constant `AWS_CUSTOM_MEMORY_MANAGEMENT`.

> If you use a version of the library that is built without the compile-time constant, global memory system functions such as `InitializeAWSMemorySystem` won't work; the global `new` and `delete` functions are used instead.

For more information about the compile-time constant, see STL and AWS Strings and Vectors.

Allocating and Deallocating Memory

To allocate or deallocate memory

1. Subclass `MemorySystemInterface`: `aws/core/utils/memory/MemorySystemInterface.h`.

```
class MyMemoryManager : public Aws::Utils::Memory::MemorySystemInterface
{
public:
    // ...
    virtual void* AllocateMemory(
        std::size_t blockSize, std::size_t alignment,
        const char *allocationTag = nullptr) override;
    virtual void FreeMemory(void* memoryPtr) override;
};
```

> **Note**
>
> You can change the type signature for `AllocateMemory` as needed.

2. Install a memory manager with an instance of your subclass by calling `InitializeAWSMemorySystem`. This should occur at the beginning of your application. For example, in your `main()` function:

```
int main(void)
{
  MyMemoryManager sdkMemoryManager;
  Aws::Utils::Memory::InitializeAWSMemorySystem(sdkMemoryManager);
  // ... do stuff
  Aws::Utils::Memory::ShutdownAWSMemorySystem();
  return 0;
}
```

3. Just before exit, call `ShutdownAWSMemorySystem` (as shown in the preceding example, but repeated here):

```
Aws::Utils::Memory::ShutdownAWSMemorySystem();
```

STL and AWS Strings and Vectors

When initialized with a memory manager, the AWS SDK for C++ defers all allocation and deallocation to the memory manager. If a memory manager doesn't exist, the SDK uses global new and delete.

If you use custom STL allocators, you must alter the type signatures for all STL objects to match the allocation policy. Because STL is used prominently in the SDK implementation and interface, a single approach in the SDK would inhibit direct passing of default STL objects into the SDK or control of STL allocation. Alternately, a hybrid approach—using custom allocators internally and allowing standard and custom STL objects on the interface—could potentially make it more difficult to investigate memory issues.

The solution is to use the memory system's compile-time constant `AWS_CUSTOM_MEMORY_MANAGEMENT` to control which STL types the SDK uses.

If the compile-time constant is enabled (on), the types resolve to STL types with a custom allocator connected to the AWS memory system.

If the compile-time constant is disabled (off), all `Aws::*` types resolve to the corresponding default `std::*` type.

Example code from the ``AWSAllocator.h`` file in the SDK

```
#ifdef AWS_CUSTOM_MEMORY_MANAGEMENT

template< typename T >
class AwsAllocator : public std::allocator< T >
{
   ... definition of allocator that uses AWS memory system
};

#else

template< typename T > using Allocator = std::allocator<T>;

#endif
```

In the example code, the `AwsAllocator` can be a custom allocator or a default allocator, depending on the compile-time constant.

Example code from the ``AWSVector.h`` file in the SDK

```
template<typename T> using Vector = std::vector<T, Aws::Allocator<T>>;
```

In the example code, we define the `Aws::*` types.

If the compile-time constant is enabled (on), the type maps to a vector using custom memory allocation and the AWS memory system.

If the compile-time constant is disabled (off), the type maps to a regular `std::vector` with default type parameters.

Type aliasing is used for all `std::` types in the SDK that perform memory allocation, such as containers, string streams, and string buffers. The AWS SDK for C++ uses these types.

Remaining Issues

You can control memory allocation in the SDK; however, STL types still dominate the public interface through string parameters to the model object `initialize` and `set` methods. If you don't use STL and use strings and containers instead, you have to create a lot of temporaries whenever you want to make a service call.

To remove most of the temporaries and allocation when you make service calls using non-STL, we have implemented the following:

- Every Init/Set function that takes a string has an overload that takes a `const char*`.
- Every Init/Set function that takes a container (map/vector) has an add variant that takes a single entry.
- Every Init/Set function that takes binary data has an overload that takes a pointer to the data and a `length` value.
- (Optional) Every Init/Set function that takes a string has an overload that takes a non-zero terminated `const char*` and a `length` value.

Native SDK Developers and Memory Controls

Follow these rules in the SDK code:

- Don't use `new` and `delete`; use `Aws::New<>` and `Aws::Delete<>` instead.
- Don't use `new[]` and `delete[]`; use `Aws::NewArray<>` and `Aws::DeleteArray<>`.
- Don't use `std::make_shared`; use `Aws::MakeShared`.
- Use `Aws::UniquePtr` for unique pointers to a single object. Use the `Aws::MakeUnique` function to create the unique pointer.
- Use `Aws::UniqueArray` for unique pointers to an array of objects. Use the `Aws::MakeUniqueArray` function to create the unique pointer.
- Don't directly use STL containers; use one of the `Aws::` typedefs or add a typedef for the container you want. For example:

```
Aws::Map<Aws::String, Aws::String> m_kvPairs;
```

- Use `shared_ptr` for any external pointer passed into and managed by the SDK. You must initialize the shared pointer with a destruction policy that matches how the object was allocated. You can use a raw pointer if the SDK is not expected to clean up the pointer.

Logging

The AWS SDK for C++ includes logging support that you can configure. When initializing the logging system, you can control the filter level and the logging target (file with a name that has a configurable prefix or a stream). The log file generated by the prefix option rolls over once per hour to allow for archiving or deleting log files.

```
Aws::Utils::Logging::InitializeAWSLogging(
    Aws::MakeShared<Aws::Utils::Logging::DefaultLogSystem>(
        "RunUnitTests", Aws::Utils::Logging::LogLevel::TRACE, "aws_sdk_"));
```

If you don't call `InitializeAWSLogging` in your program, the SDK will not do any logging. If you do use logging, don't forget to shut it down at the end of your program by calling `ShutdownAWSLogging`:

```
Aws::Utils::Logging::ShutdownAWSLogging();
```

Example integration test with logging

```
#include <aws/external/gtest.h>

#include <aws/core/utils/memory/stl/AWSString.h>
#include <aws/core/utils/logging/DefaultLogSystem.h>
#include <aws/core/utils/logging/AWSLogging.h>

#include <iostream>

int main(int argc, char** argv)
{
    Aws::Utils::Logging::InitializeAWSLogging(
        Aws::MakeShared<Aws::Utils::Logging::DefaultLogSystem>(
            "RunUnitTests", Aws::Utils::Logging::LogLevel::TRACE, "aws_sdk_"));
    ::testing::InitGoogleTest(&argc, argv);
    int exitCode = RUN_ALL_TESTS();
    Aws::Utils::Logging::ShutdownAWSLogging();
    return exitCode;
}
```

Error Handling

The AWS SDK for C++ does not use exceptions; however, you can use exceptions in your code. Every service client returns an outcome object that includes the result and an error code.

Example of handling error conditions

```
bool CreateTableAndWaitForItToBeActive()
{
  CreateTableRequest createTableRequest;
  AttributeDefinition hashKey;
  hashKey.SetAttributeName(HASH_KEY_NAME);
  hashKey.SetAttributeType(ScalarAttributeType::S);
  createTableRequest.AddAttributeDefinitions(hashKey);
  KeySchemaElement hashKeySchemaElement;
  hashKeySchemaElement.WithAttributeName(HASH_KEY_NAME).WithKeyType(KeyType::HASH);
  createTableRequest.AddKeySchema(hashKeySchemaElement);
  ProvisionedThroughput provisionedThroughput;
  provisionedThroughput.SetReadCapacityUnits(readCap);
  provisionedThroughput.SetWriteCapacityUnits(writeCap);
  createTableRequest.WithProvisionedThroughput(provisionedThroughput);
  createTableRequest.WithTableName(tableName);
```

```
  CreateTableOutcome createTableOutcome = dynamoDbClient->CreateTable(createTableRequest);
  if (createTableOutcome.IsSuccess())
  {
     DescribeTableRequest describeTableRequest;
     describeTableRequest.SetTableName(tableName);
     bool shouldContinue = true;
     DescribeTableOutcome outcome = dynamoDbClient->DescribeTable(describeTableRequest);

     while (shouldContinue)
     {
         if (outcome.GetResult().GetTable().GetTableStatus() == TableStatus::ACTIVE)
         {
            break;
         }
         else
         {
            std::this_thread::sleep_for(std::chrono::seconds(1));
         }
     }
     return true;
  }
  else if(createTableOutcome.GetError().GetErrorType() == DynamoDBErrors::RESOURCE_IN_USE)
  {
     return true;
  }

  return false;
}
```

Working with AWS Services

This section provides guidance and tips for working with particular AWS services.

IAM

This section provides examples of programming IAM using the AWS SDK for C++.

> ### Note
>
> Only the code that is necessary to demonstrate each technique is supplied here, but complete example code is available on GitHub, where you can download a single source file or you can clone the repository locally to get all examples, build and run them.

Managing IAM Users

> ### Note
>
> These code snippets assume that you understand the material in *Getting Started* and have configured default AWS credentials using the information in *Providing AWS Credentials*.

Create a User

Use the IAMClient CreateUser function, passing it a CreateUserRequest with the name of the user to create.

Includes:

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/CreateUserRequest.h>
#include <aws/iam/model/CreateUserResult.h>
```

Code:

```
Aws::IAM::IAMClient iam;
Aws::IAM::Model::CreateUserRequest create_request;
create_request.SetUserName(user_name);

auto create_outcome = iam.CreateUser(create_request);
if(!create_outcome.IsSuccess()) {
    std::cout << "Error creating IAM user " << user_name << ":" <<
        create_outcome.GetError().GetMessage() << std::endl;
```

```
    return;
}
```

This call will fail if the user already exists. You can avoid this by first verifying if the user exists or not by calling the IAMClient GetUser function. The function will fail with `Aws::IAM::IAMErrors::NO_SUCH_ENTITY` if the user doesn't already exist.

Includes:

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/GetUserRequest.h>
#include <aws/iam/model/GetUserResult.h>
```

Code:

```
Aws::IAM::IAMClient iam;
Aws::IAM::Model::GetUserRequest get_request;
get_request.SetUserName(user_name);

auto get_outcome = iam.GetUser(get_request);
if(get_outcome.IsSuccess()) {
    std::cout << "IAM user " << user_name << " already exists" << std::endl;
    return;
} else if (get_outcome.GetError().GetErrorType() !=
        Aws::IAM::IAMErrors::NO_SUCH_ENTITY) {
    std::cout << "Error checking existence of IAM user " << user_name << ":"
        << get_outcome.GetError().GetMessage() << std::endl;
    return;
}
```

See the complete example.

Listing Users

List the existing IAM users for your account by calling the IAMClient ListUsers function, passing it a ListUsersRequest object. The list of users is returned in a ListUsersResult object that you can use to get information about the users.

The result may be paginated; to check to see if there are more results available, check the value of `GetResult().GetIsTruncated()`. If `true`, then set a marker on the request and call ListUsers again to get the next batch of users. This code demonstrates the technique.

Includes:

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/ListUsersRequest.h>
#include <aws/iam/model/ListUsersResult.h>
```

Code:

```
Aws::IAM::IAMClient iam;
Aws::IAM::Model::ListUsersRequest request;

bool done = false;
bool header = false;
while (!done)
{
    auto outcome = iam.ListUsers(request);
    if (!outcome.IsSuccess()) {
        std::cout << "Failed to list iam users:" <<
            outcome.GetError().GetMessage() << std::endl;
        break;
    }

    if (!header) {
        std::cout << std::left << std::setw(32) << "Name" <<
            std::setw(30) << "ID" << std::setw(64) << "Arn" <<
            std::setw(20) << "CreateDate" << std::endl;
        header = true;
    }

    const auto &users = outcome.GetResult().GetUsers();
    for (const auto &user : users) {
        std::cout << std::left << std::setw(32) << user.GetUserName() <<
            std::setw(30) << user.GetUserId() << std::setw(64) <<
            user.GetArn() << std::setw(20) <<
            user.GetCreateDate().ToGmtString(DATE_FORMAT) << std::endl;
    }

    if (outcome.GetResult().GetIsTruncated()) {
        request.SetMarker(outcome.GetResult().GetMarker());
    } else {
        done = true;
    }
}
```

See the complete example.

Update a User

To update an existing user, create an UpdateUserRequest and pass it to the IAMClient UpdateUser member function.

Includes:

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/UpdateUserRequest.h>
```

Code:

```
Aws::IAM::Model::UpdateUserRequest request;
request.SetUserName(old_name);
request.SetNewUserName(new_name);

auto outcome = iam.UpdateUser(request);
if (outcome.IsSuccess()) {
    std::cout << "IAM user " << old_name <<
        " successfully updated with new user name " << new_name <<
        std::endl;
} else {
    std::cout << "Error updating user name for IAM user " << old_name <<
        ":" << outcome.GetError().GetMessage() << std::endl;
}
```

See the complete example.

Delete a User

To delete an existing user, call the IAMClient DeleteUser function, passing it a DeleteUserRequest object containing the name of the user to delete.

Includes:

```
#include <aws/core/Aws.h>
#include <aws/iam/IAMClient.h>
#include <aws/iam/model/DeleteUserRequest.h>
```

Code:

```
Aws::IAM::IAMClient iam;
Aws::IAM::Model::DeleteUserRequest request;
request.SetUserName(user_name);
auto outcome = iam.DeleteUser(request);
if(!outcome.IsSuccess()) {
    std::cout << "Error deleting IAM user " << user_name << ": " <<
        outcome.GetError().GetMessage() << std::endl;
    return;
}
std::cout << "Successfully deleted IAM user " << user_name << std::endl;
```

See the complete example.

Amazon S3

This section provides examples of programming Amazon S3 using the AWS SDK for C++.

> ### Note
>
> Only the code that is necessary to demonstrate each technique is supplied here, but complete example code is available on GitHub, where you can download a single source file or you can clone the repository locally to get all examples, build and run them.

Creating, Listing, and Deleting Buckets

Every object (file) in Amazon Simple Storage Service must reside within a *bucket*, which represents a collection (container) of objects. Each bucket is known by a *key* (name), which must be unique. For detailed information about buckets and their configuration, see Working with Amazon S3 Buckets in the *Amazon S3 Developer Guide*.

> **Best Practice**
>
> We recommend that you enable the AbortIncompleteMultipartUpload lifecycle rule on your Amazon S3 buckets.
>
> This rule directs Amazon S3 to abort multipart uploads that don't complete within a specified number of days after being initiated. When the set time limit is exceeded, Amazon S3 aborts the upload and then deletes the incomplete upload data.
>
> For more information, see Lifecycle Configuration for a Bucket with Versioning in the *Amazon S3 User Guide*.

> ### Note
>
> These code snippets assume that you understand the material in *Getting Started* and have configured default AWS credentials using the information in *Providing AWS Credentials*.

Create a Bucket

Use the S3Client object `CreateBucket` method, passing it a `CreateBucketRequest` with the bucket's name.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/CreateBucketRequest.h>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::CreateBucketRequest bucket_request;
    bucket_request.WithBucket(bucket_name);

    auto create_bucket_outcome = s3_client.CreateBucket(bucket_request);

    if (create_bucket_outcome.IsSuccess()) {
        std::cout << "Done!" << std::endl;
    } else {
        std::cout << "CreateBucket error: " <<
            create_bucket_outcome.GetError().GetExceptionName() << std::endl
            << create_bucket_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

List Buckets

Use the S3Client object `ListBucket` method. If successful, the method returns a `ListBucketOutcome` object, which contains a `ListBucketResult` object.

Use the `ListBucketResult` object `GetBuckets` method to get a list of `Bucket` objects that contain information about each Amazon S3 bucket in your account.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/Bucket.h>
```

Code

```
{
    Aws::S3::S3Client s3_client;
    auto list_buckets_outcome = s3_client.ListBuckets();

    if (list_buckets_outcome.IsSuccess()) {
        std::cout << "Your Amazon S3 buckets:" << std::endl;

        Aws::Vector<Aws::S3::Model::Bucket> bucket_list =
            list_buckets_outcome.GetResult().GetBuckets();

        for (auto const &bucket: bucket_list) {
            std::cout << "* " << bucket.GetName() << std::endl;
        }
```

```
    } else {
        std::cout << "ListBuckets error: " <<
            list_buckets_outcome.GetError().GetExceptionName() << " " <<
            list_buckets_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

Delete a Bucket

Use the S3Client object `DeleteBucket` method, passing it a `DeleteBucketRequest` object that is set with the name of the bucket to delete. *The bucket must be empty or an error will result*.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/DeleteBucketRequest.h>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::DeleteBucketRequest bucket_request;
    bucket_request.WithBucket(bucket_name);

    auto delete_bucket_outcome = s3_client.DeleteBucket(bucket_request);

    if (delete_bucket_outcome.IsSuccess()) {
        std::cout << "Done!" << std::endl;
    } else {
        std::cout << "DeleteBucket error: " <<
            delete_bucket_outcome.GetError().GetExceptionName() << std::endl
            << delete_bucket_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

Operations on Objects

An Amazon S3 object represents a *file*, which is a collection of data. Every object must reside within a *bucket*.

> **Note**
>
> These code snippets assume that you understand the material in *Getting Started* and have configured default AWS credentials using the information in *Providing AWS Credentials*.

Upload an Object

Use the S3Client object `PutObject` method, supplying it with a bucket name, key name, and file to upload. *The bucket must exist or an error will result*.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/PutObjectRequest.h>
#include <iostream>
#include <fstream>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::PutObjectRequest object_request;
    object_request.WithBucket(bucket_name).WithKey(key_name);

    // Binary files must also have the std::ios_base::bin flag or'ed in
    auto input_data = Aws::MakeShared<Aws::FStream>("PutObjectInputStream",
            file_name.c_str(), std::ios_base::in);

    object_request.SetBody(input_data);

    auto put_object_outcome = s3_client.PutObject(object_request);

    if (put_object_outcome.IsSuccess()) {
        std::cout << "Done!" << std::endl;
    } else {
        std::cout << "PutObject error: " <<
            put_object_outcome.GetError().GetExceptionName() << " " <<
            put_object_outcome.GetError().GetMessage() << std::endl;
```

```
    }
}
```

See the complete example.

List Objects

To get a list of objects within a bucket, use the S3Client object `ListObjects` method. Supply it with a `ListObjectsRequest` that you set with the name of a bucket to list the contents of.

The `ListObjects` method returns a `ListObjectsOutcome` object that you can use to get a list of objects in the form of `Object` instances.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/ListObjectsRequest.h>
#include <aws/s3/model/Object.h>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::ListObjectsRequest objects_request;
    objects_request.WithBucket(bucket_name);

    auto list_objects_outcome = s3_client.ListObjects(objects_request);

    if (list_objects_outcome.IsSuccess()) {
        Aws::Vector<Aws::S3::Model::Object> object_list =
            list_objects_outcome.GetResult().GetContents();

        for (auto const &s3_object: object_list) {
            std::cout << "* " << s3_object.GetKey() << std::endl;
        }
    } else {
        std::cout << "ListObjects error: " <<
            list_objects_outcome.GetError().GetExceptionName() << " " <<
            list_objects_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

Download an Object

Use the S3Client object `GetObject` method, passing it a `GetObjectRequest` that you set with the name of a bucket and the object key to download. `GetObject` returns a `GetObjectOutcome` object that you can use to access the S3 object's data.

The following example downloads an object from Amazon S3 and saves its contents to a file (using the same name as the object's key).

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/GetObjectRequest.h>
#include <fstream>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::GetObjectRequest object_request;
    object_request.WithBucket(bucket_name).WithKey(key_name);

    auto get_object_outcome = s3_client.GetObject(object_request);

    if (get_object_outcome.IsSuccess()) {
        Aws::OFStream local_file;
        local_file.open(key_name.c_str(), std::ios::out | std::ios::binary);
        local_file << get_object_outcome.GetResult().GetBody().rdbuf();
        std::cout << "Done!" << std::endl;
    } else {
        std::cout << "GetObject error: " <<
            get_object_outcome.GetError().GetExceptionName() << " " <<
            get_object_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

Delete an Object

Use the S3Client object's `DeleteObject` method, passing it a `DeleteObjectRequest` that you set with the name of a bucket and object to download. *The specified bucket and object key must exist or an error will result*.

Includes

```
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/DeleteObjectRequest.h>
#include <fstream>
```

Code

```
{
    Aws::S3::S3Client s3_client;

    Aws::S3::Model::DeleteObjectRequest object_request;
    object_request.WithBucket(bucket_name).WithKey(key_name);

    auto delete_object_outcome = s3_client.DeleteObject(object_request);

    if (delete_object_outcome.IsSuccess()) {
        std::cout << "Done!" << std::endl;
    } else {
        std::cout << "DeleteObject error: " <<
            delete_object_outcome.GetError().GetExceptionName() << " " <<
            delete_object_outcome.GetError().GetMessage() << std::endl;
    }
}

Aws::ShutdownAPI(options);
```

See the complete example.

Amazon SQS Examples

This section provides examples of programming Amazon SQS using the AWS SDK for C++.

Note

Only the code that is necessary to demonstrate each technique is supplied here, but complete example code is available on GitHub, where you can download a single source file or you can clone the repository locally to get all examples, build and run them.

Working with Amazon SQS Message Queues

A *message queue* is the logical container you use to send messages reliably in Amazon SQS. There are two types of queues: *standard* and *first-in, first-out* (FIFO). To learn more about queues and the differences between these types, see the *Amazon SQS Developer Guide*.

These C++ examples show you how to use the AWS SDK for C++ to create, list, delete, and get the URL of an Amazon SQS queue.

Create a Queue

Use the SQSClient class `CreateQueue` member function, and provide it with a CreateQueueRequest object that describes the queue parameters.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/CreateQueueRequest.h>
#include <aws/sqs/model/CreateQueueResult.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::CreateQueueRequest cq_req;
cq_req.SetQueueName(queue_name);

auto cq_out = sqs.CreateQueue(cq_req);
if (cq_out.IsSuccess()) {
    std::cout << "Successfully created queue " << queue_name << std::endl;
} else {
    std::cout << "Error creating queue " << queue_name << ": " <<
    cq_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

List Queues

To list Amazon SQS queues for your account, call the SQSClient class `ListQueues` member function, and pass it a ListQueuesRequest object.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
#include <aws/sqs/model/ListQueuesResult.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::ListQueuesRequest lq_req;

auto lq_out = sqs.ListQueues(lq_req);
if (lq_out.IsSuccess()) {
    std::cout << "Queue Urls:" << std::endl << std::endl;
```

```
    const auto &queue_urls = lq_out.GetResult().GetQueueUrls();
    for (const auto &iter : queue_urls) {
        std::cout << " " << iter << std::endl;
    }
} else {
    std::cout << "Error listing queues: " <<
        lq_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

Get the Queue's URL

To get the URL for an existing Amazon SQS queue, call the SQSClient class `GetQueueUrl` member function.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/GetQueueUrlRequest.h>
#include <aws/sqs/model/GetQueueUrlResult.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::GetQueueUrlRequest gqu_req;
gqu_req.SetQueueName(queue_name);

auto gqu_out = sqs.GetQueueUrl(gqu_req);
if (gqu_out.IsSuccess()) {
    std::cout << "Queue " << queue_name << " has url " <<
    gqu_out.GetResult().GetQueueUrl() << std::endl;
} else {
    std::cout << "Error getting url for queue " << queue_name << ": " <<
        gqu_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

Delete a Queue

Provide the URL to the SQSClient class `DeleteQueue` member function.

Includes

```
#include <aws/core/Aws.h>
#include <aws/core/client/DefaultRetryStrategy.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/DeleteQueueRequest.h>
```

Code

```
Aws::Client::ClientConfiguration client_cfg;
client_cfg.retryStrategy =
    Aws::MakeShared<Aws::Client::DefaultRetryStrategy>(
        "sqs_delete_queue", 0);
Aws::SQS::SQSClient sqs(client_cfg);

Aws::SQS::Model::DeleteQueueRequest dq_req;
dq_req.SetQueueUrl(queue_url);

auto dq_out = sqs.DeleteQueue(dq_req);
if (dq_out.IsSuccess()) {
    std::cout << "Successfully deleted queue with url " << queue_url <<
        std::endl;
} else {
    std::cout << "Error deleting queue " << queue_url << ": " <<
        dq_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

More Info

- How Amazon SQS Queues Work in the *Amazon SQS Developer Guide*
- CreateQueue in the *Amazon SQS API Reference*
- GetQueueUrl in the *Amazon SQS API Reference*
- ListQueues in the *Amazon SQS API Reference*
- DeleteQueues in the *Amazon SQS API Reference*

Sending, Receiving, and Deleting Amazon SQS Messages

Messages are always delivered using an *SQS queue*. These C++ examples show you how to use the AWS SDK for C++ to send, receive, and delete Amazon SQS messages from SQS queues.

Send a Message

You can add a single message to an Amazon SQS queue by calling the SQSClient class `SendMessage` member function. You provide `SendMessage` with a SendMessageRequest object containing the queue's URL, the message body, and an optional delay value (in seconds).

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/SendMessageRequest.h>
#include <aws/sqs/model/SendMessageResult.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::SendMessageRequest sm_req;
sm_req.SetQueueUrl(queue_url);
sm_req.SetMessageBody(msg_body);

auto sm_out = sqs.SendMessage(sm_req);
if (sm_out.IsSuccess()) {
    std::cout << "Successfully sent message to " << queue_url <<
        std::endl;
} else {
    std::cout << "Error sending message to " << queue_url << ": " <<
        sm_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

Receive Messages

Retrieve any messages that are currently in the queue by calling the SQSClient class `ReceiveMessage` member function, passing it the queue's URL. Messages are returned as a list of Message objects.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ReceiveMessageRequest.h>
#include <aws/sqs/model/ReceiveMessageResult.h>
```

Code

```
Aws::Client::ClientConfiguration client_cfg;
client_cfg.requestTimeoutMs = 30000;

Aws::SQS::SQSClient sqs(client_cfg);

Aws::SQS::Model::ReceiveMessageRequest rm_req;
rm_req.SetQueueUrl(queue_url);
rm_req.SetMaxNumberOfMessages(1);

auto rm_out = sqs.ReceiveMessage(rm_req);
if (!rm_out.IsSuccess()) {
    std::cout << "Error receiving message from queue " << queue_url << ": "
        << rm_out.GetError().GetMessage() << std::endl;
    return;
}

const auto& messages = rm_out.GetResult().GetMessages();
if (messages.size() == 0) {
    std::cout << "No messages received from queue " << queue_url <<
```

```
        std::endl;
    return;
}

const auto& message = messages[0];
std::cout << "Received message:" << std::endl;
std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() << std::endl;
std::cout << "  Body: " << message.GetBody() << std::endl << std::endl;
```

See the complete example.

Delete Messages after Receipt

After receiving a message and processing its contents, delete the message from the queue by sending the message's receipt handle and the queue URL to the SQSClient class `DeleteMessage` member function.

Includes

```
#include <aws/sqs/model/DeleteMessageRequest.h>
```

Code

```
Aws::SQS::Model::DeleteMessageRequest dm_req;
dm_req.SetQueueUrl(queue_url);
dm_req.SetReceiptHandle(message.GetReceiptHandle());

auto dm_out = sqs.DeleteMessage(dm_req);
if (dm_out.IsSuccess()) {
    std::cout << "Successfully deleted message " << message.GetMessageId()
        << " from queue " << queue_url << std::endl;
} else {
    std::cout << "Error deleting message " << message.GetMessageId() <<
        " from queue " << queue_url << ": " <<
        dm_out.GetError().GetMessage() << std::endl;
}
```

See the complete example.

More Info

- How Amazon SQS Queues Work in the *Amazon SQS Developer Guide*
- SendMessage in the *Amazon SQS API Reference*
- SendMessageBatch in the *Amazon SQS API Reference*
- ReceiveMessage in the *Amazon SQS API Reference*
- DeleteMessage in the *Amazon SQS API Reference*

Enabling Long Polling for Amazon SQS Message Queues

Amazon SQS uses *short polling* by default, querying only a subset of the servers—based on a weighted random distribution—to determine whether any messages are available for inclusion in the response.

Long polling helps reduce your cost of using Amazon SQS by reducing the number of empty responses when there are no messages available to return in reply to a ReceiveMessage request sent to an Amazon SQS queue and eliminating false empty responses.

> ### Note
>
> You can set a long polling frequency from *1–20 seconds*.

Enabling Long Polling when Creating a Queue

To enable long polling when creating an Amazon SQS queue, set the `ReceiveMessageWaitTimeSeconds` attribute on the CreateQueueRequest object before calling the SQSClient class' `CreateQueue` member function.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/CreateQueueRequest.h>
#include <aws/sqs/model/CreateQueueResult.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::CreateQueueRequest request;
request.SetQueueName(queue_name);
request.AddAttributes(
    Aws::SQS::Model::QueueAttributeName::ReceiveMessageWaitTimeSeconds,
    poll_time);

auto outcome = sqs.CreateQueue(request);
if (outcome.IsSuccess()) {
    std::cout << "Successfully created queue " << queue_name <<
        std::endl;
} else {
    std::cout << "Error creating queue " << queue_name << ": " <<
        outcome.GetError().GetMessage() << std::endl;
}
```

See the complete example.

Enabling Long Polling on an Existing Queue

In addition to enabling long polling when creating a queue, you can also enable it on an existing queue by setting `ReceiveMessageWaitTimeSeconds` on the SetQueueAttributesRequest before calling the SQSClient class' `SetQueueAttributes` member function.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/SetQueueAttributesRequest.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::SQS::Model::SetQueueAttributesRequest request;
request.SetQueueUrl(queue_url);
request.AddAttributes(
    Aws::SQS::Model::QueueAttributeName::ReceiveMessageWaitTimeSeconds,
    poll_time);

auto outcome = sqs.SetQueueAttributes(request);
if (outcome.IsSuccess()) {
    std::cout << "Successfully updated long polling time for queue " <<
        queue_url << " to " << poll_time << std::endl;
} else {
    std::cout << "Error updating long polling time for queue " <<
        queue_url << ": " << outcome.GetError().GetMessage() <<
        std::endl;
}
```

See the complete example.

Enabling Long Polling on Message Receipt

You can enable long polling when receiving a message by setting the wait time in seconds on the ReceiveMessageRequest that you supply to the SQSClient class' ReceiveMessage member function.

> **Note**
>
> You should make sure that the AWS client's request timeout is larger than the maximum long poll time (20s) so that your `ReceiveMessage` requests don't time out while waiting for the next poll event!

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ReceiveMessageRequest.h>
#include <aws/sqs/model/ReceiveMessageResult.h>
```

Code

```
Aws::Client::ClientConfiguration client_cfg;
client_cfg.requestTimeoutMs = 30000;

Aws::SQS::SQSClient sqs(client_cfg);

Aws::SQS::Model::ReceiveMessageRequest request;
request.SetQueueUrl(queue_url);
request.SetMaxNumberOfMessages(1);
request.SetWaitTimeSeconds(wait_time);

auto outcome = sqs.ReceiveMessage(request);
```

See the complete example.

More Info

- Amazon SQS Long Polling in the *Amazon SQS Developer Guide*
- CreateQueue in the *Amazon SQS API Reference*
- ReceiveMessage in the *Amazon SQS API Reference*
- SetQueueAttributes in the *Amazon SQS API Reference*

Setting Visibility Timeout in Amazon SQS

When a message is received in Amazon SQS, it remains on the queue until it's deleted in order to ensure receipt. A message that was received, but not deleted, will be available in subsequent requests after a given *visibility timeout* to help prevent the message from being received more than once before it can be processed and deleted.

Note

When using standard queues, visibility timeout isn't a guarantee against receiving a message twice. If you are using a standard queue, be sure that your code can handle the case where the same message has been delivered more than once.

Setting the Message Visibility Timeout upon Message Receipt

When you have received a message, you can modify its visibility timeout by passing its receipt handle in a ChangeMessageVisibilityRequest that you pass to the SQSClient class' `ChangeMessageVisibility` member function.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ChangeMessageVisibilityRequest.h>
#include <aws/sqs/model/ReceiveMessageRequest.h>
```

Code

```
Aws::Client::ClientConfiguration client_config;
client_config.requestTimeoutMs = 30000;

Aws::SQS::SQSClient sqs(client_config);

Aws::SQS::Model::ReceiveMessageRequest receive_request;
receive_request.SetQueueUrl(queue_url);
receive_request.SetMaxNumberOfMessages(1);

auto receive_outcome = sqs.ReceiveMessage(receive_request);
if (!receive_outcome.IsSuccess()) {
    std::cout << "Error receiving message from queue " << queue_url << ": "
        << receive_outcome.GetError().GetMessage() << std::endl;
    return;
}

const auto& messages = receive_outcome.GetResult().GetMessages();
if (messages.size() == 0) {
    std::cout << "No messages received from queue " << queue_url <<
        std::endl;
    return;
}

const auto& message = messages[0];
std::cout << "Received message:" << std::endl;
std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() << std::endl;
std::cout << "  Body: " << message.GetBody() << std::endl << std::endl;

Aws::SQS::Model::ChangeMessageVisibilityRequest request;
request.SetQueueUrl(queue_url);
request.SetReceiptHandle(message.GetReceiptHandle());
request.SetVisibilityTimeout(visibility_timeout);
auto outcome = sqs.ChangeMessageVisibility(request);
if (outcome.IsSuccess()) {
```

```
    std::cout << "Successfully changed visibility of message " <<
        message.GetMessageId() << " from queue " << queue_url << std::endl;
} else {
    std::cout << "Error changing visibility of message " <<
        message.GetMessageId() << " from queue " << queue_url << ": " <<
        outcome.GetError().GetMessage() << std::endl;
}
```

See the complete example.

More Info

- Visibility Timeout in the *Amazon SQS Developer Guide*
- SetQueueAttributes in the *Amazon SQS API Reference*
- GetQueueAttributes in the *Amazon SQS API Reference*
- ReceiveMessage in the *Amazon SQS API Reference*
- ChangeMessageVisibility in the *Amazon SQS API Reference*
- ChangeMessageVisibilityBatch in the *Amazon SQS API Reference*

Using Dead Letter Queues in Amazon SQS

Amazon SQS provides support for *dead letter queues*. A dead letter queue is a queue that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead letter queue to determine why their processing did not succeed.

To create a dead letter queue, you must first create a *redrive policy*, and then set the policy in the queue's attributes.

> ### Important
>
> A dead letter queue must be the same type of queue (FIFO or standard) that the source queue is. It must also be created using the same AWS account and region as the source queue.

Creating a Redrive Policy

A redrive policy is specified in JSON. To create it, you can use the JSON utility class provided with the AWS SDK for C++.

Here is an example function that creates a redrive policy by providing it with the ARN of your dead letter queue and the maximum number of times the message can be received and not processed before it's sent to the dead letter queue.

Includes

```
#include <aws/core/Aws.h>
#include <aws/core/utils/json/JsonSerializer.h>
```

Code

```
Aws::String MakeRedrivePolicy(const Aws::String& queue_arn, int max_msg)
{
    Aws::Utils::Json::JsonValue redrive_arn_entry;
    redrive_arn_entry.AsString(queue_arn);

    Aws::Utils::Json::JsonValue max_msg_entry;
    max_msg_entry.AsInteger(max_msg);

    Aws::Utils::Json::JsonValue policy_map;
    policy_map.WithObject("deadLetterTargetArn", redrive_arn_entry);
    policy_map.WithObject("maxReceiveCount", max_msg_entry);

    return policy_map.WriteReadable();
}
```

See the complete example.

Setting the Redrive Policy on your Source Queue

To finish setting up your dead letter queue, call the SQSClient class' `SetQueueAttributes` member function with a SetQueueAttributesRequest object for which you've set the `RedrivePolicy` attribute with your JSON redrive policy.

Includes

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/SetQueueAttributesRequest.h>
```

Code

```
Aws::SQS::SQSClient sqs;

Aws::String redrivePolicy = MakeRedrivePolicy(queue_arn, max_msg);

Aws::SQS::Model::SetQueueAttributesRequest request;
request.SetQueueUrl(src_queue_url);
request.AddAttributes(
    Aws::SQS::Model::QueueAttributeName::RedrivePolicy,
    redrivePolicy);

auto outcome = sqs.SetQueueAttributes(request);
if (outcome.IsSuccess()) {
    std::cout << "Successfully set dead letter queue for queue  " <<
```

```
    src_queue_url << " to " << queue_arn << std::endl;
} else {
    std::cout << "Error setting dead letter queue for queue " <<
    src_queue_url << ": " << outcome.GetError().GetMessage() <<
    std::endl;
}
```

More Info

- Using Amazon SQS Dead Letter Queues in the *Amazon SQS Developer Guide*
- SetQueueAttributes in the *Amazon SQS API Reference*

Document History

This topic lists major changes to the *AWS SDK for C++ Developer Guide* over the course of its history.

- **Latest documentation update:** Aug 02, 2018

March 10, 2017

- Added new topics to *Amazon SQS Examples*:
  - *Using Dead Letter Queues in Amazon SQS*
  - *Enabling Long Polling for Amazon SQS Message Queues*
  - *Setting Visibility Timeout in Amazon SQS*

February 27, 2017

- A new topic in the **Getting Started** section, *Using the SDK*, has been added to show how to properly initialize and shutdown the SDK.
- In addition to the existing *Amazon S3* examples, new code examples have been added for *Amazon SQS Examples* and *IAM*.

February 02, 2016

Documentation first created.

About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model: you are charged only for the services that you—or your applications—use. For new AWS users, a free usage tier is available. On this tier, services are free below a certain level of usage. For more information about AWS costs and the Free Tier, see Use the AWS Free Tier. To obtain an AWS account, visit the AWS home page and click **Create a Free Account**.