AWS SDK for C++ Developer Guide

Release 1.0

Amazon Web Services

October 29, 2016

CHAPTER 1

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 is meant to be performant and fully functioning with low-level and high-level SDKs, while minimizing dependencies and providing platform portability (Windows, OS X, Linux, and mobile).

1.1 Additional Documentation and Resources

For more valuable online resources that are available for AWS SDK for C++ developers, see the following:

- 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
- License

CHAPTER 2

Getting Started

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

2.1 Setting Up the SDK

2.1.1 Prerequisites

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

- Visual Studio 2013 or later

  Note: Visual Studio 2013 does not 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 may 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*, and *libuuid*. Typically, you'll find the packages in your system's package manager.

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

```
sudo apt-get install libcurl-dev libssl-dev uuid-dev
```

On *Fedora-based systems*, use:

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

2.1.2 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 website 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 you chose 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.

2.1.3 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 allows 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
```

   - Clone with Git (SSH):

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

2. Install cmake and the relevant build tools for your platform. Ensure these are available in your `PATH`. If you are unable to install **`cmake`**, you can use **`make`** or **`msbuild`**.
3. Create a directory to create the buildfiles in, and generate the necessary buildfiles within it (referred to as an *out-of-source build*, the recommended approach):

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

Alternatively, you can 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:

   - For auto make systems:

```
make
sudo make install
```

   - For 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 information about additional 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 have the appropriate environment variables (`ANDROID_NDK`) set.

Android on Windows

Building for Android on Windows requires additional setup. In particular, you have to run **cmake** from a Visual Studio developer command prompt (2013 or later). You'll also need the commands **git** and **patch** in your path. If you have git installed on a Windows system, then **patch** is likely found 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 serial fashion. To make things quicker, we recommend installing JOM as an alternative to **nmake** and then changing the **cmake** invocation to:

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

Creating Release Builds

To create a *release* build of the SDK, do one of the following:

- For auto make systems:

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

- For 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 building your project, you can run these executables to ensure everything works correctly.

2.2 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++ can be done 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, OS X, 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, OS X, 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.

Once 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.

2.3 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++.

2.3.1 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. Enter the directory and add a `CMakeLists.txt` file that specifies your project name, its executables, sourcefiles, 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
    add_executable(my-example my-example.cpp)

    # The libraries used by your executable.
    # "aws-cpp-sdk-s3" is just an example.
    target_link_libraries(my-example aws-cpp-sdk-s3)
```

Note: There are many options that you can set in your `CMakeLists.txt` build configuration file. For an introduction to the file's features, see the tutorial on the CMake website.

2.3.2 (Optional) Setting CMAKE_PREFIX_PATH

CMake needs to know the location of the `aws-sdk-cpp-config.cmake` so that it can properly resolve the AWS SDK libraries that your application uses. This file can be found in the build directory that you used to build the SDK.

By setting the path in `CMAKE_PREFIX_PATH`, you won't need to type this path every time you rebuild your application.

You can set it on Linux, OS X, or Unix like this:

```
export CMAKE_PREFIX_PATH=/path/to/sdk_build_dir
```

On Windows, use `set` instead:

```
set CMAKE_PREFIX_PATH=C:\path\to\sdk_build_dir
```

2.3.3 Building with CMake

To build your application with **cmake**, create a directory to build into:

```
mkdir my_project_build
```

Enter the directory and run **cmake** with the path to your project's source directory:

```
cd my_project_build
cmake ../my_example_project
```

If you did not set `CMAKE_PREFIX_PATH`, then you must add the path to the SDK's build directory using `-Daws-sdk-cpp_DIR`:

```
cmake -Daws-sdk-cpp_DIR=/path/to/sdk_build_dir ../my_example_project
```

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

```
make
```

CHAPTER 3

Configuring the SDK

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

3.1 CMake Parameters

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

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

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

3.1.1 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=someOther
```

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*

3.1.2 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*
- *ANDROID_NATIVE_API_LEVEL*
- *ANDROID_STL*
- *ANDROID_TOOLCHAIN_NAME*
- *DISABLE_ANDROID_STANDALONE_BUILD*
- *NDK_DIR*

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.

3.2 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;
};
```

3.2.1 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 cooresponds 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 Simple Storage Service 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`.

3.3 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.

3.4 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 Simple Storage Service `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);
```

CHAPTER 4

Programming with the SDK

This section provides information about general use of the AWS SDK for C++. For service-specific examples, see Working with AWS Services.

4.1 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`*`::`*`Service`*`Client`.

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.

4.1.1 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_TA

// 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);
```

4.1.2 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);
```

4.1.3 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);
```

4.2 Utility Modules

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

4.2.1 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.

4.2.2 String Utils

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

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

4.2.3 Hashing Utils

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

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

4.2.4 JSON Parser

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

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

4.2.5 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.

4.3 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*.

4.3.1 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();
```

4.3.2 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.

4.3.3 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.

4.3.4 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.

4.4 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();
```

Here is an 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;
}
```

4.5 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;
}
```

CHAPTER 5

Working with AWS Services

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

5.1 Amazon S3

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

Note: Only the code needed to demonstrate each technique is supplied here. However, complete example code is available on GitHub, where you can download a single source file or you can clone the repository locally to view, build, and run all of the examples.

5.1.1 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 `credentials`.

Create a Bucket

Use the S3Client object's `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;
}
```

See the complete example.

List Buckets

Use the S3Client object's `ListBucket` method. If successful, a `ListBucketOutcome` object is returned, which contains a `ListBucketResult` object.

Use the `ListBucketResult` object's `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;
}
```

See the complete example.

Delete a Bucket

Use the S3Client object's `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;
}
```

See the complete example.

5.1.2 Operations on Objects

An Amazon S3 object represents a *file*, or 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 `credentials`.

Upload an Object

Use the S3Client object's `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:

```
const Aws::String bucket_name = argv[1];
const Aws::String key_name = argv[2];
const Aws::String dir_name = ".";

std::cout << "Uploading " << key_name << " to S3 bucket: " <<
    bucket_name << std::endl;

Aws::S3::S3Client s3_client;

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

auto input_data = Aws::MakeShared<Aws::FStream>(key_name.c_str(), dir_name.c_str(), std::i

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's `ListObjects` method, supplying it with a `ListObjectsRequest` that you set with the name of a bucket to list the contents of.

The `ListObjects` method returns an `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:

```
const Aws::String bucket_name = argv[1];
std::cout << "Objects in S3 bucket: " << bucket_name << std::endl;

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;
}
```

See the complete example.

Download an Object

Use the S3Client object's `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 can be used 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:

```
const Aws::String bucket_name = argv[1];
const Aws::String key_name = argv[2];

std::cout << "Downloading " << key_name << " from S3 bucket: " <<
    bucket_name << std::endl;

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;
}
```

See the complete example.

Delete an Object

Use the S3Client object's `DeleteObject` method, passing it a `DeleteObjectRequest` which 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:

```
const Aws::String bucket_name = argv[1];
const Aws::String key_name = argv[2];

std::cout << "Deleting" << key_name << " from S3 bucket: " <<
    bucket_name << std::endl;

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;
}
```

See the complete example.

CHAPTER 6

Document History

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

- **Latest documentation update:** October 29, 2016

February 02, 2016 Documentation first created.

CHAPTER 7

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**.

Index