# API Middleware Overview

Middleware is software that's assembled into an API’s ‘app pipeline’ to handle requests and responses. Each component chooses whether to pass the request to the next component in the pipeline and can perform work before and after the next component in the pipeline is invoked.

Request delegates are used to build the request pipeline. The request delegates handle each HTTP request.

Request delegates are configured using Run, Map, and Use extension methods. An individual request delegate can be specified in-line as an anonymous method (called in-line middleware), or it can be defined in a reusable class. These reusable classes and in-line anonymous methods are middleware, also called middleware components. Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline or short-circuiting the pipeline.

API Middleware delegates various connection tasks as data traverses the API. This allows a single thread the handle multiple in-line middleware or reusable classes.

API Middleware is a mainstay of ASP.NET modules and HTTP handlers, but is also found in other API tools and in links between non-.Net services and ASP.Net services.

ASP.NET Core Middleware examples include UseMVC, UseStaticFiles, and UseIdentity. Public APIs like Twitter, Google, and Facebook provide developers with an application key; naming them as “app-key”, “user-key”, “api-key” and so on.

HTTP and Managed File Transfer are the most common API Middleware tools.

## An API Artifact is Not

An artifact in the context of an API Middleware function is not the supporting documentation, workflow, security verification or any manmade file. It is solely the resulting text or binary object produced by an API function, that is stored or moved as a result of that function.

# API Artifacts

“API Artifact" is the term used to describe something tangible (i.e. a file) that is produced; in this context as a byproduct when running a software function. Thus, the runnables/libs are the artifacts produced when compiling source, and the image is the artifact produced by the whole "build" step, which is an artifact containing one or more other artifacts

An API Artifact can be anything created during the process of running an API, or one of its middleware subcomponents.

An API Artifact can be as big as an entire system image, libraries, or executables created as a Docker Container when moving a system, or as small as a URL referencing the pathway to a file, or as simple as a text string containing a few numbers that unlock a function or store an IP address.

An artifact may be encrypted, plain text or a binary file.

# Types of Artifacts produced by an API

An API function usually produces an expectedset of artifacts based on set of API functions approved and developed for use between systems.

For instance, an API whose function is to do file transfers with the Managed File Transfer tool will create an artifact that is a binary image of the file being transferred. If that file is encrypted, it will have a separate artifact that contains the PKI key or other key to decrypt the file. Encrypted artifacts and their keys typically reside in separate locations, with separate paths.

The most common API artifacts are:

* Managed File Transfer (MFT) binary and encrypted binary files
* Keystores
* Data table
* Templates
* Scripts,
* Synapse configurations and custom sequences etc.
* Configuration files
* Registry data
* User management data
* Files profile changes,
* Webapps
* Execution plans
* Event receivers
* Indexing files
* Node ID
* Shard allocation related data
* Key stores
* Datasources
* User management data
* Application specific and proprietary; data, calls, scripts and files

# API Naming Conventions

### For the VA we are providing Vendor agnostic and Mulesoft specific naming conventions.

## Vendor Agnostic API Naming Convention

### Middleware Artifacts Naming Convention

Middleware artifacts are named based on the type of an artifact and its purpose. Different middleware platforms have various types of artifacts but most of them can be grouped into the following categories:

* Connectors
* Flows
* Tasks
* Custom objects and methods
* Configurations

Connectors are platform-provided or custom integrations to third-party platforms. For example, a common connector is a database connector for a specific database platform such as Oracle or SQL Server. Another example is a Salesforce connector. Connector naming convention varies with each middleware platform, and each platform has its best practice standards. As a best practice, name the connector after the platform it is connecting to. For example, “MySalesforceConnector”.

Flows and sub-flows are groupings of connectors and tasks in a logical workflow. A flow name should indicate the purpose and type of a flow. If this is an API-based flow, it is recommended to have the word “api” in the flow name. The word “flow” should be at the end of the flow name. Typically, flow names are all lower-cased, with words concatenated with a dash character. For example, “customer-api-flow” or “address-sub-flow”. Batch mode flows and routines should have the word “batch” in the name to indicate that this flow or sub-flow is not triggered via a REST or SOAP API, but rather is a scheduled batch process. For example, “account-sync-batch-flow”.

Tasks and routines should contain the key operation in the name. For example, if the task is to call a web service, the name should contain callWebService or similar. Task or operation names are usually lower-cased, following Hungarian notation, similar to how Java methods are typically named.

Custom objects and methods should follow the objects and methods naming convention as described in the previous section of this document.

Finally, configuration files follow the naming convention based on the main programming language used within a specific middleware platform. As most of the middleware platforms are built on Java programming language, except for Microsoft BizTalk, the configuration files and folders should be named following Java file naming conventions.

## MuleSoft-specific API Naming Convention

### Middleware Artifacts Naming Convention

Mulesoft middleware artifacts are separated into the following:

* Flows
* Sub-flows
* Connectors
* Mule configuration files

Mule flow and sub-flow names should be all lower-cased, with words separated by dashes. API-based flows should contain the word “api”. All flow names should end with the word “flow”. All sub-flow names should end with a word “sub-flow”. Batch flows should contain the word “batch” in the name of the flow or sub-flow. For example, “customer-apiFlow” or “account-sync-batchFlow”.

In addition, the flows and sub-flows that are tied to a specific platform, such as Anypoint MQ, should have “mq” next to the word “flow”. For example, “account-sync-mqFlow”.

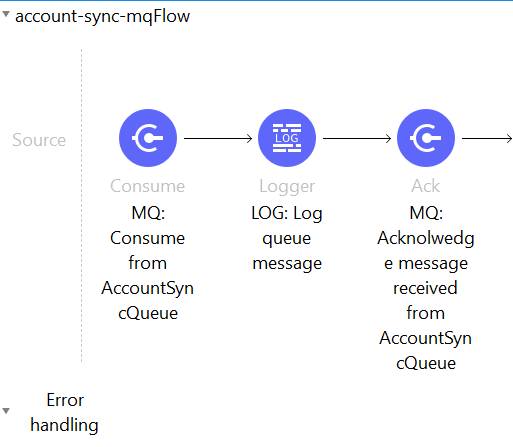
The Mule connectors, as they are named in the flows, should indicate the source system that they are used to connect to. For example, a Mulesoft Salesforce Connector should have the target Salesforce instance in the name of the connector.

It is common to also label specific connectors such as Logging or Anypoint MQ with an all-CAPS prefix:

* “LOG:” for a default logging connector
* “JSON:” for a JSON logging connector
* “MQ:” for an Anypoint MQ connector
* “SFDC:” for a Salesforce connector
* “DB:” for a database connector

For example, “LOG: Log Message Payload” to log the message payload.

A screenshot below shows how the connector names are defined in a sample Mule flow.



Mulesoft configuration files follow the following naming convention:

* All files are lower-cased
* Words are concatenated with dashes in the file names

For example, “mule-app.propertes” or “pom.xml”.