# What is a Reusable Artifact

## Vendor-agnostic Reusable Artifact

Reusable artifacts are created to externalize and package reusable processes, routines, error handlers, and components across more than one API. For example, a reusable validation routine with complex business rules can be packaged into a reusable artifact, to be used by multiple APIs that rely on the same business rules and validation. Relying on reusable artifacts during API development provides the following benefits:

* Enforcement of development standards
* Code reuse and sharing across the teams
* Standard formatting and messaging across the APIs
* Reduction of code complexity
* Better collaboration across the teams

For example, having a reusable artifact that addresses consistent logging of API incoming requests and responses provides standard audit logging of payloads across all APIs within the application. Having a reusable artifact that handles all of the API errors and exceptions provides a common solution to error handling and exception messaging across all of the APIs. Capturing complex validation rules for an application into a shared module provides a reusable validation component that can be used across all of the APIs that require such validation rules within an application.

Reusable artifacts for API development can be represented in many forms, depending on the programming language and an API development platform used. For example, a reusable artifact can be represented as a shared .NET or Java component, a NodeJS module, a Python shared script, or a Javascript include file with reusable functions. Typically, a reusable API artifact is represented as a common integration component (e.g., a custom backend platform connector), transformation process, validation routine, shared logger or error handler.

## MuleSoft-specific Reusable Artifact

### What is a Reusable Artifact for Mulesoft Development

#### Reusable Artifacts

There are several needs for shared components that arise when developing Mule applications. A typical use case is to package those flows and sub-flows that are reusable across multiple Mule process applications into a shared component. Another use case is when a backend platform requires a custom shared connector for all application flows and sub-flows to access from within the same Mule application or series of APIs. Yet another scenario is when a set of reusable validation logic and business rules is packaged as a reusable module to be included into Mule process flows. Finally, with Mule 4.x, shared reusable transformations and functions can now be packaged into .dw files, to be imported inside the DataWeave transformations, in the header section of the transformation definition.

These scenarios require creating reusable shared artifacts within Mule, whether it is a shared library, connector, module, or function, to be shared across Mule applications for consistency and code reuse.

#### Shared API Commons Library

Mule 3 and 4.x provide ability to package shared flows, error handlers, and configuration files, as shared reusable libraries. These libraries are packaged as .jar files and imported either into Anypoint Exchange or shared component repository such as JFrog Artifactory or Maven private repository. The libraries can then be imported into any Mule application that requires the use of that library, by adding the shared library .jar as a Maven dependency in the pom.xml.

#### Shared Connectors and Modules

To connect to a backend platform that requires custom integration logic, there may be a need to develop and deploy a custom connector. If there are no connectors on Anypoint Exchange already developed to meet the needs of this requirement, a custom connector is recommended. Once the connector is created and published to Exchange, any Mule project that needs to leverage this connector can now add it to the project, by reviewing the list of connectors on the organization’s Exchange dashboard.

In some cases, there may be a need to create a custom module with reusable logic such as custom validation or security integration, without a need to connect to the backend source system. In that case, a custom Mule module should be created. Once the shared module or connector is packaged, it should be added to the organization’s Anypoint Exchange repository.

#### Reusable Transformations

A new feature of Mule 4.x allows a developer of a reusable transformation or function used within the transformation to externalize that function or transformation into a separate .dw file. The file can then be imported in the header declaration of any transformation across other Mule applications, and therefore reused without re-writing the functions repeatedly.