# **Practical Exercise: Writing a Custom Handler**

Training Objective

Learn how to write a custom handler and add it to an API.

Business Scenario

PizzaShack has a requirement to rate limit requests based on the IP address.

High-Level Steps

* Write custom handler
* Engage handler to API

Detailed Instructions

**Before you begin**: Create an API with a back-end in order to test the functionality. The sample API that is shipped with the product can also be used for this purpose.

Write Custom Handler

Write a custom throttle handler to rate limit requests based on IP address. Use the sample code shared in [Writing a custom handler](https://apim.docs.wso2.com/en/4.2.0/develop/extending-api-manager/extending-gateway/writing-custom-handlers/) and modify as necessary.

# **Writing Custom Handlers**

This section introduces handlers and explains how to write a custom handler using an example.

## **Introducing Handlers**

When an API is created and deployed, the API is added to the API Gateway Memory. It has a set of handlers, each of which is executed on the APIs in the same order they appear in the configuration. You can find the default handlers in any API's Synapse definition as shown below if you enable Synapse artifacts file storage.

<**handlers**>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.common.APIMgtLatencyStatsHandler">

<**property** name="apiUUID" value="<apiUUID>"/>

</**handler**>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.security.CORSRequestHandler">

<**property** name="apiImplementationType" value="ENDPOINT"/>

<**property** name="AuthorizationHeader" value="Authorization"/>

</**handler**>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.common.APIStatusHandler"/>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler">

<**property** name="RemoveOAuthHeadersFromOutMessage" value="true"/>

<**property** name="APILevelPolicy" value=""/>

<**property** name="AuthorizationHeader" value="Authorization"/>

<**property** name="keyManagers" value="all"/>

<**property** name="CertificateInformation" value="{}"/>

<**property** name="APISecurity" value="oauth2,oauth\_basic\_auth\_api\_key\_mandatory"/>

<**property** name="apiUUID" value="<apiUUID>"/>

</**handler**>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.throttling.ThrottleHandler"/>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.analytics.APIMgtGoogleAnalyticsTrackingHandler">

<**property** name="configKey" value="ga-config-key"/>

</**handler**>

<**handler** class="org.wso2.carbon.apimgt.gateway.handlers.ext.APIManagerExtensionHandler"/>

</**handlers**>

Let's see what each handler does:

* **APIMgtLatencyStatsHandler:** Publishes request and response latencies, if analytics is enabled.
* **CORSRequestHandler:** Sets the CORS headers to the request and executes the CORS sequence mediation logic. This handler is thereby responsible for returning the CORS headers from the gateway or routing the requests to the backend and letting the backend send the CORS headers.
* **APIStatusHandler:** Handles the status of APIs. For instance this handler mediates blocked APIs from blocked sequence.
* **APIAuthenticationHandler:** Validates the OAuth2 bearer token used to invoke the API. It also determines whether the token is of type Production or Sandbox and sets MessageContext variables as appropriate.
* **ThrottleHandler:** Throttles requests based on the throttling policy specified by the policyKey property. Throttling is applied both at the application level as well as subscription level.
* **APIMgtGoogleAnalyticsTrackingHandler:** Publishes events to Google Analytics. This handler only comes into effect If Google analytics tracking is enabled. See Integrating with Google Analytics for more information.
* **APIManagerExtensionHandler** : Triggers extension sequences. By default, the extension handler is listed at last in the handler chain, and therefore is executed last. You cannot change the order in which the handlers are executed, except the extension handler. To configure the API Gateway to execute extension handler first, log in to management console https://localhost:9443/carbon and in the main tab, expand the resources section and browse for \_system/config/apimgt/applicationdata/tenant-conf.json. Edit the field ExtensionHandlerPosition and provide the value top. This is useful when you want to execute your own extensions before our default handlers in situations like doing additional security checks such as signature verification on access tokens before executing the default security handler. See [Adding Mediation Extensions](https://apim.docs.wso2.com/en/latest/deploy-and-publish/deploy-on-gateway/api-gateway/message-mediation/changing-the-default-mediation-flow-of-api-requests).

### **Using APILogMessageHandler**

Message logging is handled by APIManagerExtensionHandler. APILogMessageHandler is a sample handler that comes with WSO2 API Manager that can be used for logging.

**Info**

**Why are logs removed from APIManagerExtensionHandler?**

The primary purpose of ExtensionHandler is handling extensions to mediation and not for logging messages. When the logs are also included in ExtensionHandler, there's a limitation to improve the ExtensionHandler for developing features because it breaks the logs.

For example, When the ExtensionHandler moves to the top of the handlers set, most of the logs print null values since the handler runs before the APIAuthenticationHandler. Therefore, the logs are removed from the extension handler and APILogMessageHandler introduced as a sample.

**Note**

To achieve logging requirements, this handler is not the only approach and with custom sequences also it is possible to log messages using the Log Mediator.

In order to enable logging by invoking APILogMessageHandler, follow the steps below.

**To enable Message Logging into APIS created from publisher automatically:**

1. Open the <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file and copy the following handler before </Handlers>.

<handler **class**="org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageHandler"/>

**Note**

In a fully distributed setup, this configuration should be done in the Traffic manager Node.

1. Copy the following code into the <APIM\_HOME>/repository/conf/log4j2.properties file to enable printing DEBUG logs.

logger.log-msg-handler.name = org.wso2.carbon.apimgt.gateway.handlers.logging.APILogMessageHandler

logger.log-msg-handler.level = DEBUG

Append the log-msg-handler logger name to loggers configuration which is a comma-separated list of all active loggers. The sample configuration is given below.

loggers = log-msg-handler, trace-messages, org-apache-coyote,com-hazelcast

**Note**

The logger name log-msg-handler can be replaced by any logger-name.

1. Restart API Manager.

## **Writing a custom handler**

**Note**

The outcomes of using Class Mediator vs Custom Handler are very similar. However, when using a custom handler you need to maintain a customized velocity template file that needs to be manually merged when you upgrade your product to a newer version. Therefore, it is recommended to use custom Handlers when you wish to specify the exact order of execution of JARs as this can not be done with [Mediators](https://apim.docs.wso2.com/en/latest/deploy-and-publish/deploy-on-gateway/api-gateway/message-mediation/changing-the-default-mediation-flow-of-api-requests).

Custom Handler is a way of extending API Manager which the product offer to change the API flow through the API Gateway. What is happening in custom handler can be decided by the code you are writing to extend the product. It can be adding extra security, logging database invocation or something else. This custom handler must extend the org.apache.synapse.rest.AbstractHandler class and implement handleRequest() and handleResponse() methods.

Let's see how you can write a custom handler and apply it to the API Manager. In this example, we extend the authentication handler. Make sure your custom handler name is not the same as the name of an existing handler.

WSO2 API Manager provides the OAuth2 bearer token as its default authentication mechanism. A sample implementation is [here](https://apim.docs.wso2.com/en/latest/assets/attachments/learn/api-authentication-handler.java). Similarly, you can extend the API Manager to support any custom authentication mechanism by writing your own authentication handler class.

Given below is an example implementation. Please find the complete project archive [here](https://apim.docs.wso2.com/en/latest/assets/attachments/reference/org.wso2.carbon.test.authenticator.zip). You can download, unzip and build the project using maven and Java 7 or 8.

**package** org.wso2.carbon.test;

**import** org.apache.synapse.MessageContext;

**import** org.apache.synapse.core.axis2.Axis2MessageContext;

**import** org.apache.synapse.rest.AbstractHandler;

**import** org.wso2.carbon.apimgt.gateway.handlers.security.APISecurityException;

**import** java.util.Map;

**public** **class** CustomAPIAuthenticationHandler **extends** AbstractHandler {

**public** **boolean** handleRequest(MessageContext messageContext) {

**try** {

**if** (authenticate(messageContext)) {

**return** **true**;

}

} **catch** (APISecurityException e) {

e.printStackTrace();

}

**return** **false**;

}

**public** **boolean** handleResponse(MessageContext messageContext) {

**return** **true**;

}

**public** **boolean** authenticate(MessageContext synCtx) **throws** APISecurityException {

Map headers = getTransportHeaders(synCtx);

String authHeader = getAuthorizationHeader(headers);

**if** (authHeader.startsWith("userName")) {

**return** **true**;

}

**return** **false**;

}

**private** String getAuthorizationHeader(Map headers) {

**return** (String) headers.get("Authorization");

}

**private** Map getTransportHeaders(MessageContext messageContext) {

**return** (Map) ((Axis2MessageContext) messageContext).getAxis2MessageContext().

getProperty(org.apache.axis2.context.MessageContext.TRANSPORT\_HEADERS);

}

}

Make sure to update the pom file for the above project you created (or downloaded) with below dependency.

<**dependencies**>

<**dependency**>

<**groupId**>org.apache.synapse</**groupId**>

<**artifactId**>synapse-core</**artifactId**>

<**version**>4.0.0-wso2v20</**version**>

<**scope**>provided</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>org.apache.synapse</**groupId**>

<**artifactId**>synapse-commons</**artifactId**>

<**version**>4.0.0-wso2v20</**version**>

<**scope**>provided</**scope**>

</**dependency**>

<**dependency**>

<**groupId**>org.wso2.carbon.apimgt</**groupId**>

<**artifactId**>org.wso2.carbon.apimgt.gateway</**artifactId**>

<**version**>9.28.112</**version**>

<**scope**>provided</**scope**>

</**dependency**>

</**dependencies**>

## **Engaging the custom handler**

1. Build the custom authenticator code created/downloaded previously.
2. If the jar you created is non-OSGI, you should place the jar in the <APIM\_HOME>/repository/components/lib directory otherwise you should place the jar in the <APIM\_HOME>/repository/components/dropins driectory.
3. Engage the custom handler using the API template as explained below: You can engage a custom handler to all APIs at once or only to selected APIs.

To engage a custom handler to all APIs, you need to add the custom handler with its logic in the <APIM\_HOME>/repository/resources/api\_templates/velocity\_template.xml file. For example, the following code segment adds the custom authentication handler that you wrote earlier to the velocity\_template.xml file while making sure that it skips the default APIAuthenticationHandler implementation:

<handlers xmlns="http://ws.apache.org/ns/synapse">

<handler **class**="org.wso2.carbon.test.CustomAPIAuthenticationHandler" />

#foreach($handler in $handlers)

#**if**(!($handler.className == "org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"))

<handler xmlns="http://ws.apache.org/ns/synapse" **class**="$handler.className">

#**if**($handler.hasProperties())

#set ($map = $handler.getProperties() )

#foreach($property in $map.entrySet())

<property name="$!property.key" value="$!property.value"/>

#end

#end

</handler>

#end

#end

</handlers>

To engage a custom handler only to selected APIs, you can add a condition. Given below is an example of adding only the CustomAPIAuthenticationHandler to the sample PizzaShackAPI.

<**handlers** xmlns="http://ws.apache.org/ns/synapse">

#if($apiName == 'admin--PizzaShackAPI')

<**handler** class="org.wso2.carbon.sample.auth.CustomAPIAuthenticationHandler"/>

#end

#foreach($handler in $handlers)

#if($apiName != 'admin--PizzaShackAPI' || !($handler.className == "org.wso2.carbon.apimgt.gateway.handlers.security.APIAuthenticationHandler"))

<**handler** xmlns="http://ws.apache.org/ns/synapse" class="$handler.className">

#if($handler.hasProperties())

#set ($map = $handler.getProperties() )

#foreach($property in $map.entrySet())

<**property** name="$!property.key" value="$!property.value"/>

#end

#end

</**handler**>

#end

#end

</**handlers**>

1. Restart the API Manager server.

Expected Outcome

When the API is called, rate limiting will differ according to the IP address.