
Esquio

Release 0.0.1

Jul 29, 2020

1	Terminology	3
1.1	Product	3
1.2	Feature	3
1.3	Toggle	3
1.4	Parameter	3
1.5	Store	4
2	Contributing	5
2.1	How to contribute?	5
2.2	General feedback and discussions?	5
2.3	Bugs and feature requests?	5
2.4	Contribution projects	5
3	Getting started with Esquio in .NET Core	7
3.1	Create a folder for your new project	7
3.2	Create the project	7
3.3	Installation	7
3.4	Setup	8
3.5	Test the application	9
4	Getting started with Esquio in ASP.NET Core	11
4.1	Setup	11
4.2	ASP.NET Core Web Apps	12
4.3	ASP.NET Core MVC	13
4.4	SPA and Native Apps	14
5	Getting started with Esquio Azure DevOps tasks	15
5.1	Setup	15
5.2	Prerequisites	15
5.3	Create Esquio API key	15
5.4	Setup Esquio Service Connection	16
5.5	Esquio rollout task	18
5.6	Esquio rollback task	19
5.7	Esquio set toggle parameter task	20
6	Behaviors	23
6.1	OnErrorBehavior	23

6.2	NotFoundBehavior	23
7	Esquio toggles	25
7.1	UserNameToggle	25
7.2	RoleNameToggle	25
7.3	EnvironmentToggle	26
7.4	FromToToggle	26
7.5	GradualRolloutUserNameToggle	27
8	Esquio ASP.NET Core toggles	29
8.1	ClaimValueToggle	29
8.2	GradualRolloutClaimValueToggle	30
8.3	GradualRolloutHeaderValueToggle	30
8.4	GradualRolloutSessionToggle	31
9	Extensibility	33
9.1	Creating your custom toggle	33
10	Esquio CLI	37
11	Esquio & MiniProfiler	39
11.1	Setup	40



Esquio is a [Feature Toggles \(aka Feature Flags\)](#) and A/B testing library for ASP.NET Core 3.0. Feature toggling is a powerful technique that allows developers to deliver new functionality to users without changing code. Feature toggles provide an alternative to maintaining multiple branches (aka feature branches), so any feature can be tested even before it is completed and ready for the release. We can release a version of our product without production-ready features. These non production-ready features are hidden (toggled) for the broader set of users but can be enabled to any subset of testing or internal users we want to try out the features. We can even use feature toggling to enable or disable features during runtime.

Esquio is built with the possibility of use it not only in ASP.NET Core 3.0 in mind, but making it possible to use also in other .NET Core 3.0 projects like workers, webjobs, classlibraries, ... almost any kind of .NET Core 3.0 project. For the Esquio team, this is not only about using a library, but using a full Feature Toggles framework for all of our projects, and as a delivery mechanism.

We believe Feature Toggling is, somekind, a way of delivering software, making it a first class citizen in your DevOps processes, therefore we are working hard towards integrating it, via extension and pipelines tasks, with Azure DevOps, so you can use Esquio Toggles directly in your releases and delivery flows. Having a full toggle delivery experience.

Esquio Azure DevOps extensions are built in top of the Esquio API, in the case you need to integrate Esquio with any other tool, you can always use this API to handle the toggles.

Additionally, if you need it, Esquio has a full UI developed, so you can be able to handle all your Toggles in it, making it fairly simple to use and manage.

The documentation and object model use a certain terminology that you should be aware of.

1.1 Product

Allows you to manage multiple different software projects, for example, one solution can contains a web application and windows application that need the same set of features. Each product has its own unique set of features.

1.2 Feature

Features are characteristics of your product that describe its appearance, components, and capabilities. A feature is a slice of business functionality that has a corresponding benefit or set of benefits for that product's end user. Each feature has its own set of toggles.

1.3 Toggle

Toggles allows you to control when a feature is enabled or not. Esquio provides many toggles out-of-the-box such us percentage rollouts, target specific users or environments, expiration dates or even hit the 'kill' switch for a feature programmatically.

1.4 Parameter

Parameters are variables that toggles need in their validation process.

1.5 Store

A mechanism to allow you to store persistent the Esquio's object model such as products, features, toggles, parameters. Esquio provides out of the box two stores:

- ASP.NET Core JSON Configuration Provider.
- Entity Framework Core.

We are very open to community contributions, but there are a couple of guidelines you should follow so we can handle this without too much effort.

2.1 How to contribute?

The easiest way to contribute is to open an issue and start a discussion. Then we can decide if and how a feature or a change could be implemented. If you should submit a pull request with code changes, start with a description, only make the minimal changes to start with and provide tests that cover those changes.

Also read this first: [Being a good open source citizen](#)

2.2 General feedback and discussions?

Please start a discussion on the [core repo issue tracker](#)

2.3 Bugs and feature requests?

Please log a new issue in the [GitHub repo](#)

2.4 Contribution projects

We very much appreciate if you start a contribution project (e.g. support for Toogle X or Store Y). Tell us about it so we can tweet and link it in our docs.

We generally don't want to take ownership of those contribution libraries, we are already really busy supporting the core projects.

Happy coding!!!

Getting started with Esquio in .NET Core

In this article, we are going to see how easy it is to use Esquio in your .NET Core application using the NuGet packages provided by Xabaril.

> In [samples/ConsoleApp](#) you'll find a complete Esquio example in ASP.NET Core.

3.1 Create a folder for your new project

Open a command prompt and run:

```
mkdir consoleapp  
cd consoleapp
```

3.2 Create the project

To create the project type the following command using the .NET Core CLI:

```
dotnet new console
```

3.3 Installation

To install Esquio type the following command:

```
dotnet package add Microsoft.Extensions.DependencyInjection  
dotnet package add Microsoft.Extensions.Logging.Console  
dotnet package add Microsoft.Extensions.Configuration.Json  
dotnet package add Esquio  
dotnet restore
```

or using Powershell or Package Manager:

```
Install-Package Microsoft.Extensions.DependencyInjection
Install-Package Microsoft.Extensions.Logging.Console
Install-Package Microsoft.Extensions.Configuration.Json
Install-Package Esquio
```

3.4 Setup

In the Program.cs change the Main method to async:

```
static async Task Main(string[] args)
```

Import the configuration namespace:

```
using Microsoft.Extensions.Configuration;
```

Create the configuration object:

```
var configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appsettings.json")
    .Build();
```

Add a appsettings.json file to the project and add the content below:

```
{
  "Esquio": {
    "Products": [
      {
        "Name": "Console",
        "Features": [
          {
            "Name": "Colored",
            "Enabled": true,
            "Toggles": []
          }
        ]
      }
    ]
  }
}
```

Register the Esquio services and the configuration store:

```
var services = new ServiceCollection()
    .AddLogging(configure => configure.AddConsole())
    .AddEsquio()
    .AddConfigurationStore(configuration, "Esquio")
    .Services;
```

Build the service provider:

```
var serviceProvider = services.BuildServiceProvider();
```

Obtain a context for the IFeatureService:

```
var featureService = serviceProvider.GetService<IFeatureService>();
```

Check if “Colored” feature for the “Console” project is enabled and set the background color of the console to blue:

```
if (await featureService.IsEnabledAsync("Colored", "Console"))
{
    Console.BackgroundColor = ConsoleColor.Blue;
}
```

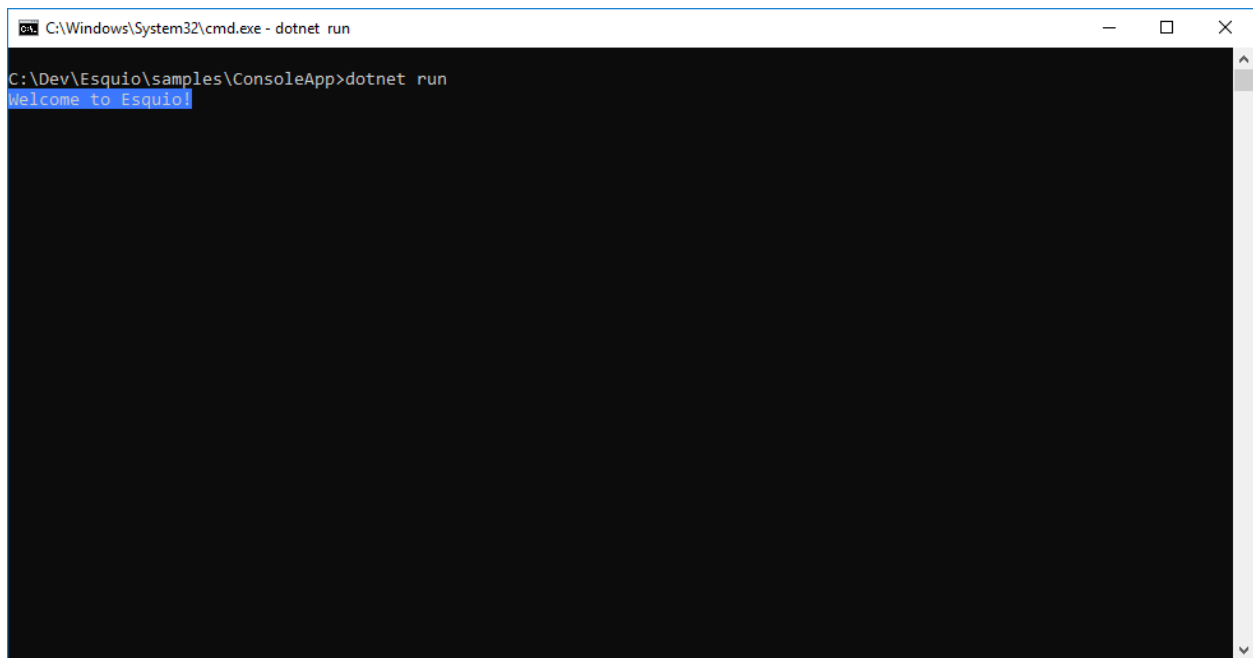
Finally write a message to check it:

```
Console.WriteLine("Welcome to Esquio!");
Console.Read();
```

3.5 Test the application

Test the app from your machine running:

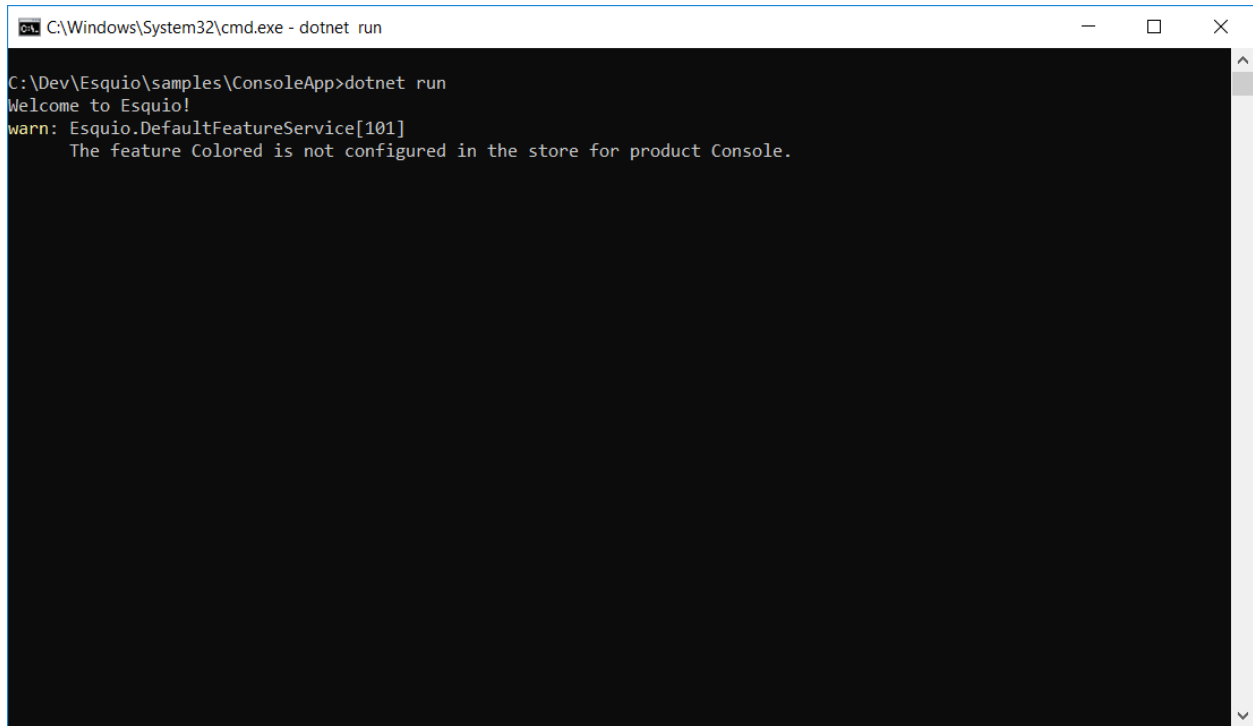
```
dotnet run
```



To disable the feature, change the `appsettings.json`:

```
"Enabled": false
```

Test again the app:



A screenshot of a Windows command prompt window. The title bar shows the path `C:\Windows\System32\cmd.exe - dotnet run`. The command prompt shows the following output:

```
C:\Dev\Esquio\samples\ConsoleApp>dotnet run
Welcome to Esquio!
warn: Esquio.DefaultFeatureService[101]
      The feature Colored is not configured in the store for product Console.
```

The output is displayed on a black background with white text. A vertical scrollbar is visible on the right side of the command prompt window.

Getting started with Esquio in ASP.NET Core

In this article, we are going to see how easy it is to use Esquio in your ASP.NET Core application using the NuGet packages provided by Xabaril.

> In [samples/WebApp](#) you'll find a complete Esquio example in ASP.NET Core.

4.1 Setup

To install Esquio open a console window and type the following command using the .NET Core CLI:

```
dotnet package add Esquio.AspNetCore
```

or using Powershell or Package Manager:

```
Install-Package Esquio.AspNetCore
```

or install via NuGet.

In the **ConfigureServices** method of Startup.cs, register the Esquio services:

```
services
    .AddEsquio()
    .AddAspNetCoreDefaultServices()
    .AddConfigurationStore(Configuration, "Esquio");
```

AddEsquio and AddAspNetCoreDefaultServices methods allows you to register the set of services that Esquio needs to works. The AddConfigurationStore method registers the configuration store to use, in this case, based on the default configuration system of [ASP.NET Core](#)

Add the content below to your appsettings.json file:

```
{
  "Esquio": {
    "Products": [
```

(continues on next page)

(continued from previous page)

```

    {
      "Name": "Console",
      "Features": [
        {
          "Name": "Colored",
          "Enabled": true,
          "Toggles": []
        }
      ]
    }
  ]
}

```

4.2 ASP.NET Core Web Apps

When working with Esquio you can attach feature metadata to an endpoint. We do this using the route mappings configuration fluent API `RequireFeature` method:

```

app.UseEndpoints(routes =>
{
    routes.MapControllerRoute(
        name: "default",
        pattern: "{controller=Match}/{action=Index}/{id?}").RequireFeature("HiddenGem");
});

```

You can specify many features separated by comma, so you can restrict access to the endpoints if a feature or a group features are enabled or not.

If you want more fine-grained control over your Controllers, Esquio provides a `FeatureFilter` attribute that forces you to supply a comma separated list of features names. You can specify that access to a controller or action method is restricted to users if these features are enabled or not:

```

[FeatureFilter(Names = Flags.MinutesRealTime)]
public IActionResult DetailLive()
{
    return View();
}

```

Also, you can use `FeatureFilter` to act as an Action constraint. You can create two Actions with the same `ActionName` and decorate one with `FeatureFilter` attribute to match the action only when the predefined feature name values are enabled or not.:

```

[ActionName("Detail")]
public IActionResult DetailWhenFlagsIsNotActive()
{
    return View();
}

[FeatureFilter(Names = Flags.MinutesRealTime)]
[ActionName("Detail")]
public IActionResult DetailWhenFlagsIsActive()
{

```

(continues on next page)

(continued from previous page)

```

return View();
}

```

Sometimes you will need to configure a fallback action. Esquio provides an `AddEndpointFallback` method that accepts a `RequestDelegate` in order to configure your custom fallback:

```

services
    .AddEsquio()
    .AddAspNetCoreDefaultServices()
    .AddConfigurationStore(Configuration, "Esquio")
    .AddEndpointFallback((context) =>
    {
        context.Response.StatusCode = StatusCodes.Status404NotFound;

        return Task.CompletedTask;
    })

```

Out-of-the-box Esquio provides `EndpointFallbackAction` class that defines common fallback actions to be used when no matching endpoints found:

- Redirect result to MVC action:

```

public static RequestDelegate RedirectToAction(string controllerName, string_
↳ actionName)

```

- Redirect result:

```

public static RequestDelegate RedirectTo(string uri)

```

- NotFound status response:

```

public static RequestDelegate NotFound()

```

4.3 ASP.NET Core MVC

With **ASP.NET MVC Core** we can use the `FeatureTagHelper` inside our Razor views to show or hide Razor fragments depending on feature is enabled or not.

```

<feature names="@Flags.MatchScore">
    <span class="badge badge-secondary badge-pill">@match.ScoreLocal - @match.
↳ ScoreVisitor</span>
</feature>

```

In this example, if the feature **MatchScore** is enabled, you can show a new design of the match score. Names property is comma-separated list of feature names to be evaluated. If any feature is not active, the tag helper will suppress the content.

The `FeatureTagHelper` supports `Include` and `Exclude` attributes:

- Include: A comma-separated list of feature names to be evaluated. If any feature is not active, this tag helper suppresses the content.
- Exclude: A comma-separated list of feature names to be evaluated. If any feature is active, this tag helper suppresses the content.

4.4 SPA and Native Apps

Single-Page-Applications and native apps are becoming the new wave for modern applications. The challenge with feature flags in these kinds of applications is handling the state transformations. In case of SPAs the changes in a webpage's DOM and the platform specific controls in native apps. We will need an endpoint to query if a feature or a set of features are enabled or not in order make real time personalization in the UX for example.

To enable this endpoint, in the `Configure` method, insert the middleware to expose the Esquio endpoint:

```
app.UseEndpoints(routes =>
{
    routes.MapEsquio(pattern: "esquio");
});
```

Now you can start your application and check out your features at `http(s)://server:port/esquio?productName=Console&featureName=Col`

```
[
{
    "enabled": true,
    "name": "Colored"
}
]
```

To disable the feature, change the `appsettings.json`:

```
"Enabled": false,
```

Test again the app:

```
[
{
    "enabled": false,
    "name": "Colored"
}
]
```

Getting started with Esquio Azure DevOps tasks

In this article, we are going to see how to configure Esquio Azure DevOps tasks for your pipelines.

> In [samples/WebApp](#) you'll find a complete Esquio example in ASP.NET Core.

5.1 Setup

The first step is install the [Esquio Azure DevOps Task from Visual Studio Marketplace](#) . There isn't anything special needed, just install it as a normal Azure DevOps extension.

Once installed, you will have new elements in your Azure DevOps:

- Esquio Service Connection
- Esquio rollout task
- Esquio rollback task
- Esquio set parameter value task

5.2 Prerequisites

To be able to use Esquio tasks, we will need to setup a Esquio [Service Connection](#) and to configure it you need to create an Esquio API key, to use it with the Esquio Service Connection.

5.3 Create Esquio API key

First of all you need to create an Esquio API key to create the [Service Connection](#) it is easily created with the Esquio UI.

Open your current deployment of Esquio UI url in a browser, and once logged-in, click on your user name in the upper right corner.



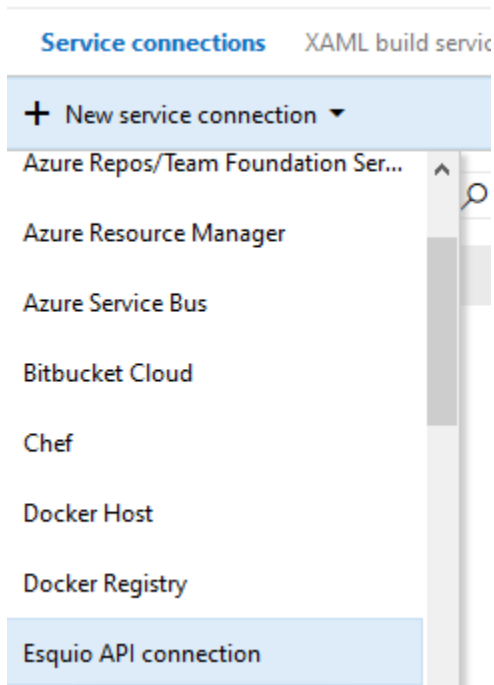
Click on *Get Private token* and you will receive a confirmation window, and your token will be copied to the clipboard, so note it to paste it when configuring the Esquio Service Connection



Now you can continue to configure the [Service Connection](#).

5.4 Setup Esquio Service Connection

[Service Connection](#) are setup per project, so open your Azure DevOps settings page, and go to Service Connections, click on *New service Connection* and select *Esquio API Connection*



This will bring the *Esquio API Connection* configuration screen, here you need to setup three parameters:

- **Connection name:** To use it in the Azure DevOps tasks.
- **Esquio API Url:** The complete url in which you have your Esquio API.
- **API token:** The Esquio API key you have setup.

×

Add Esquio API connection service connection

Connection name

Esquio API Url

ⓘ

API Token

ⓘ

[Learn More about Esquio](#)

Connection: Not verified

Verify connection

☒ Allow all pipelines to use this connection.

OK


Close

Once filled all the information, make sure it is correct, clicking on *Verify connection* and make sure it says *Connection: Verified*

Update Authentication for Esquio

Connection name	<input type="text" value="Esquio"/>	
Esquio API Url	<input type="text" value="https://esquiodemo.azurewebsites.net"/>	?
API Token	<input type="password" value="*****"/>	?

[Learn More about Esquio](#)

Connection:  Verified

[Verify connection](#)

OK

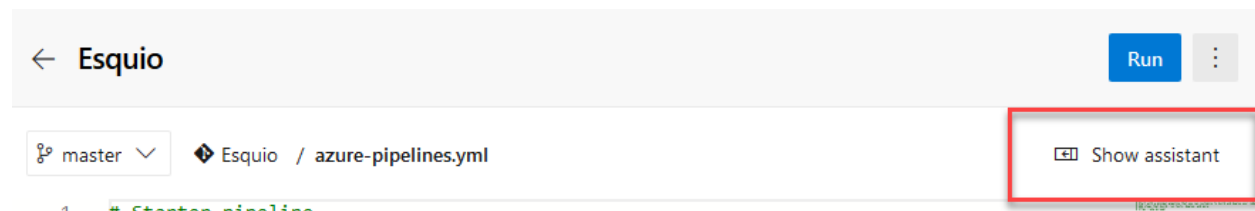
Close

Now you have setup the Esquio Connection we will need to use for the tasks.

5.5 Esquio rollout task

This task allow us to enable a feature to everyone cleaning all the toggles.

If you are creating your Azure Pipelines with *YAML* it is better to use the *YAML assistant* as it will allow you to use the datasources for the picklists.



To setup the rollout task, look for **Rollout feature with Esquio** task:

← Rollout feature with Esquio

Esquio service endpoint *

Esquio product *

Esquio feature *

We will configure three parameters:

- **Esquio Service Endpoint:** Select the previously created *Esquio Service Connection*.
- **Esquio Product:** From the list of products configured in Esquio.
- **Esquio feature:** Select, from the list of features.

The final YAML should be (with different ids) like this:

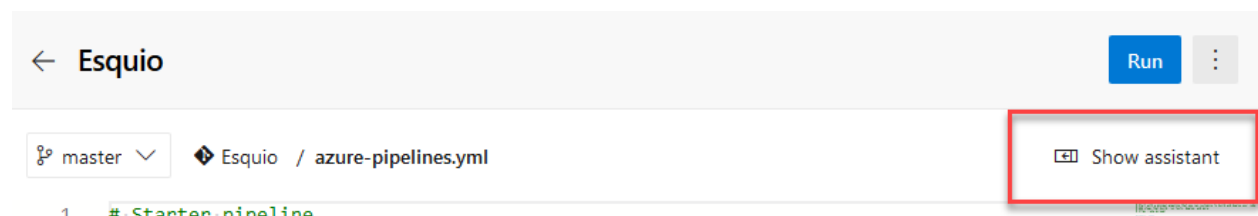
```
- task: esquio-rollout-feature@1
  inputs:
    EsquioService: 'Esquio'
    productId: '1'
    flagId: '1'
```

If you are using the classic pipelines (the visual ones), the setup is exactly the same.

5.6 Esquio rollback task

This task allow us to set a *OffToggle* for a feature, thus disabling it.

If you are creating your Azure Pipelines with *YAML* it is better to use the *YAML assistant* as it will allow you to use the datasources for the picklists.



To setup the rollout task, look for **Rollback feature with Esquio** task:

← Rollback feature with Esquio

Esquio service endpoint *

Esquio product *

Esquio feature *

We will configure three parameters:

- **Esquio Service Endpoint:** Select the previously created *Esquio Service Connection*.
- **Esquio Product:** From the list of products configured in Esquio, select the one with the feature you want to setup the *OffToggle*.
- **Esquio feature:** Select, from the list of features, the one to setup the *OffToggle*.

The final YAML should be (with different ids) like this:

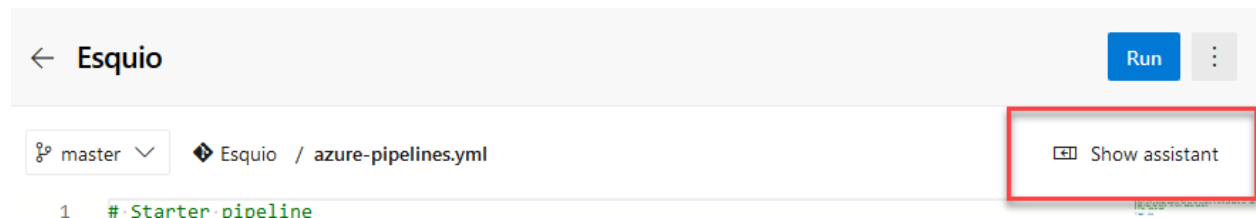
```
- task: esquio-rollback-feature@1
  inputs:
    EsquioService: 'Esquio'
    productId: '1'
    flagId: '1'
```

If you are using the classic pipelines (the visual ones), the setup is exactly the same.

5.7 Esquio set toggle parameter task

This task allow us to set a value for a particular parameter in a toggle, with this task you can setup any other type of *toggle* accepting parameters.

If you are creating your Azure Pipelines with *YAML* it is better to use the *YAML assistant* as it will allow you to use the datasources for the picklists.



To setup the rollout task, look for **Set toggle parameter with Esquio** task and select, using the picklists, the parameter for the feature toggle you want to set, and then fill-in the value you want to set for the parameter:

← Set toggle parameter with Esquio

Esquio service endpoint *

Esquio product *

Esquio feature *

Esquio Toggle *

Esquio parameter *

Esquio parameter value *

We will configure six parameters:

- **Esquio Service Endpoint:** Select the previously created *Esquio Service Connection*.
- **Esquio Product:** From the list of products configured in Esquio.
- **Esquio feature:** Select, from the list of features, the one with the parameter you want to set the value.
- **Esquio toggle:** Select, from the list of toggles, the one with the parameter you want to set the value.
- **Esquio parameter:** Select, from the list of parameters for the previously selected toggle, the one you want to set the value.
- **Esquio parameter value:** Introduce manually the value you want to setup for the parameter.

The final YAML should be (with different ids) like this:

```
- task: set-toggle-parameter@1
  inputs:
    EsquioService: 'esquio'
    productId: '1'
    flagId: '2'
    toggleId: '14'
    parameterId: 'Percentage'
    parameterValue: '59'
```

If you are using the classic pipelines (the visual ones), the setup is exactly the same.

Esquio allows you to configure their behaviors when a feature does not exists or fails during evaluation. You can configure these behaviors by modifying the Esquio setup.

6.1 OnErrorBehavior

There are three options to configure when a feature fails during evaluation:

- **OnErrorBehavior.Throw**: Re-throw the exception.
- **OnErrorBehavior.SetDisabled**: Returns disabled as a result of the evaluation.
- **OnErrorBehavior.SetEnabled**: Returns enabled as a result of the evaluation.

The `AddEsquio` method provides you a way to configure the behavior when a feature fails during evaluation:

```
services
    .AddEsquio(setup => setup.ConfigureOnErrorBehavior(OnErrorBehavior.Throw))
    .AddAspNetCoreDefaultServices()
    .AddConfigurationStore(Configuration, "Esquio");
```

In the above example, the exception will be thrown if some fail happens during the evaluation process.

6.2 NotFoundBehavior

There are three options to configure when a feature does not exists in the store:

- **NotFoundBehavior.SetDisabled**: Returns disabled as a result of the evaluation.
- **NotFoundBehavior.SetEnabled**: Returns enabled as a result of the evaluation.

The `AddEsquio` method provides you a way to configure the behavior when a feature does not exists in the store:

```
services
    .AddEsquio(setup => setup.ConfigureNotFoundBehavior(NotFoundBehavior.SetDisabled))
    .AddAspNetCoreDefaultServices()
    .AddConfigurationStore(Configuration, "Esquio");
```

In the above example, if the feature does not exist in the store, Esquio will return disabled as a result of the feature evaluation process.

A **Toggle** is code which defines when a feature is enabled or not. Each feature can use one or multiple toggles at the same time, but never more than one toggle of the same type. In Esquio you have many different toggles out of the box, and of course you can write your custom toggles.

7.1 UserNameToggle

This toggle allows you to enable features to a specific set of logged in users.

Parameters

- Users: *The collection of user(s) to activate it separated by ‘;’ character.*

```
{
  "Name": "MinutesProgressBar",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.UserNameToggle",
      "Parameters": {
        "Users": "betauser;beta"
      }
    }
  ]
}
```

7.2 RoleNameToggle

This toggle allows you to enable features to a specific set of logged in users that belong to a specific role.

Parameters

- Users: *The collection of role(s) to activate this toggle separated by ';' character.*

```
{
  "Name": "MinutesProgressBar",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.RoleNameToggle",
      "Parameters":
      {
        "Users": "betauser;beta"
      }
    }
  ]
}
```

7.3 EnvironmentToggle

This toggle allows you to enable features depending on the environment that an application is running in. This environment information is provided by [IEnvironmentNameProviderService](#). When you add Esquio to your application using `AddEsquio()` method, by default Esquio registers a [NoEnvironmentNameProviderService](#). For ASP.NET Core projects, Esquio provides a method called `AddAspNetCoreDefaultServices` that registers by default an [AspNetEnvironmentNameProviderService](#) based on `Microsoft.AspNetCore.Hosting.Abstractions.IWebHostEnvironment`.

Parameters

- Environments: *List of environments separated by semicolon you want the feature toggle to be enabled for*

```
{
  "Name": "MinutesRealTime",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.EnvironmentToggle",
      "Parameters":
      {
        "Environments": "Staging;Production"
      }
    }
  ]
}
```

7.4 FromToToggle

This toggle allows you to enable features depending on current UTC time.

Parameters

- From: *The from date (yyyy-MM-dd HH:mm:ss) interval when this toggle is activated.*
- To: *The to date (yyyy-MM-dd HH:mm:ss) interval when this toggle is activated.*

```
{
  "Name": "DarkMode",
```

(continues on next page)

(continued from previous page)

```
"Enabled": true,
"Toggles": [
  {
    "Type": "Esquio.Toggles.FromToToggle",
    "Parameters": {
      "From": "2019-06-12 00:00:00",
      "To": "2019-06-14 23:59:59"
    }
  }
]
```

7.5 GradualRolloutUserNameToggle

This toggle allows you to gradually enable features to a percentage of logged in users. Stickiness is based on the user name. Esquio uses [Jenkins hash function](#) which guarantees the user gets the same experience across many devices and also ensures that a user who is among the first 30% will also be among the first 50% of users.

Parameters

- **Percentage:** *The percentage (0-100) you want to enable the feature toggle for.*

```
{
  "Name": "DarkMode",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.GradualRolloutUserNameToggle",
      "Parameters": {
        "Percentage": 50
      }
    }
  ]
}
```

Esquio ASP.NET Core toggles

In addition to the toggles that Esquio provides out of the box, Esquio.AspNetCore provides more toggles to work with ASP.NET Core applications.

8.1 ClaimValueToggle

This toggle allows you to enable features depending on the current claims of logged in users.

Parameters

- **ClaimType**: *The claim type used to check value.*
- **ClaimValues**: *The claim value to check, multiple items separated by ';'.*

```
{
  "Name": "AnimationsMatch",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.ClaimValueToggle",
      "Parameters": {
        "ClaimType": "Company",
        "ClaimValues": "Contoso;ACME"
      }
    }
  ]
}
```

8.2 GradualRolloutClaimValueToggle

This toggle allows you gradually enabled features to a percentage of logged in users. Stickiness is based on the claim type value. Esquio uses [Jenkins hash function](#) that guarantees to the user get the same experience across many devices and also assures that a user which is among the first 30% will also be among the first 50% of the users.

Parameters

- Percentage: *The percentage (0-100) you want to enable the feature toggle for.*
- ClaimType: *The claim type used to get value to rollout.*

```
{
  "Name": "DarkMode",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.GradualRolloutClaimValueToggle",
      "Parameters": {
        "Percentage": 50,
        "ClaimType": "role"
      }
    }
  ]
}
```

8.3 GradualRolloutHeaderValueToggle

This toggle allows you gradually enabled features to a percentage of logged in users. Stickiness is based on the HTTP header value. Esquio uses [Jenkins hash function](#) that guarantees to the user get the same experience across many devices and also assures that a user which is among the first 30% will also be among the first 50% of the users.

Parameters

- Percentage: *The percentage (0-100) you want to enable the feature toggle for.*
- HeaderName: *he name of the header used to get the value to rollout.*

```
{
  "Name": "DarkMode",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.GradualRolloutHeaderValueToggle",
      "Parameters": {
        "Percentage": 50,
        "HeaderName": "X-Tenant"
      }
    }
  ]
}
```

8.4 GradualRolloutSessionToggle

This toggle allows you gradually enabled features to a percentage of user sessions (authenticated or not). Stickiness is based on the ASP.NET Core SessionId value.

Parameters

- **Percentage:** *The percentage (0-100) you want to enable the feature toggle for.*

```
{
  "Name": "DarkMode",
  "Enabled": true,
  "Toggles": [
    {
      "Type": "Esquio.Toggles.GradualRolloutSessionToggle",
      "Parameters": {
        "Percentage": 50
      }
    }
  ]
}
```


Esquio provides lot of toggles Out-of-the-box but sometimes is not enough. Therefore extensibility is a key quality attribute of any experience delivery architecture. Esquio does not aim to solve every problem but rather provides an extensible parts that enables you to adapt it to your needs.

All major components have interfaces which are extensible. Following the inversion of control design they are substitutable either through **Microsoft.Extensions.DependencyInjection** configuration.

9.1 Creating your custom toggle

Imagine that you need to create a toggle to enable features based on the user's browser (**User-Agent** header).

First step is implement the interface `IToggle`:

```
public class UserAgentBrowserToggle : IToggle
{
    public async Task<bool> IsActiveAsync(string featureName, string productName = _,
    ↪ null, CancellationToken cancellationToken = default)
    {
    }
}
```

`IsActiveAsync` returns a boolean if the feature X for the product Y is enabled or not based on this toggle. In our case, depending on the user's browser.

To be able to specify for which browsers will be this feature enabled, you need to add a property:

```
public class UserAgentBrowserToggle : IToggle
{
    public string Browsers { get; set; }

    //code omitted for brevity
}
```

If you want to an application (**Esquio UI**) be able to understand these fields and for example built an user interface in order to provide to the users a more friendly way to configure the toggles of a feature, you need to decorate the toggle with some attributes:

```
[DesignType(Description = "Toggle that is active depending on request browser_
↪information.")]
[DesignTypeParameter(ParameterName = Browsers, ParameterType = "System.String",_
↪ParameterDescription = "Collection of browser names delimited by ';' character.")]
public class UserAgentBrowserToggle : IToggle
{
    public string Browsers { get; set; }

    //code omitted for brevity
}
```

Attributes

- **DesignType**: Allow to add a friendly description for the toggle.
- **DesignTypeParameter**: Allow to add a friendly description for the toggle's parameters.

Once we have define our attributes, it's time to use the services that our toggle will need to check if is active or not. In the constructor we can specify these services:

```
public UserAgentBrowserToggle(
    IRuntimeFeatureStore featureStore,
    IHttpContextAccessor httpContextAccessor,
    ILogger<UserAgentBrowserToggle> logger)
{
    _featureStore = featureStore ?? throw new_
↪ArgumentNullException(nameof(featureStore));
    _contextAccessor = httpContextAccessor ?? throw new_
↪ArgumentNullException(nameof(httpContextAccessor));
    _logger = logger ?? throw new ArgumentNullException(nameof(logger));
}
```

Services

- **IRuntimeFeatureStore**: We use this service to retrieve the feature. Once we have the feature, we can retrieve the toggle and her data (The parameters and their values).
- **IHttpContextAccessor**: To access the HttpContext.
- **ILogger<T>**: To log whatever you want.

It's time to finish our feature. We need to complete the `IsActiveAsync` method with the code below:

```
public async Task<bool> IsActiveAsync(
    string featureName,
    string productName = null,
    CancellationToken cancellation_token = default)
{
    var feature = await _featureStore.FindFeatureAsync(featureName, productName,_
↪cancellation_token);
    var toggle = feature.GetToggle(typeof(UserAgentBrowserToggle).FullName);
    var data = toggle.GetData();

    var allowedBrowsers = data.Browsers.ToString();
    var currentBrowser = GetCurrentBrowser();
}
```

(continues on next page)

(continued from previous page)

```

    if (allowedBrowsers != null && !String.IsNullOrEmpty(currentBrowser))
    {
        _logger.LogDebug("{nameof(UserAgentBrowserToggle)} is trying to verify if
        ↳{currentBrowser} is satisfying allowed browser configuration.");

        var tokenizer = new StringTokenizer(allowedBrowsers, split_characters);

        foreach (var segment in tokenizer)
        {
            if (segment.Value?.IndexOf(currentBrowser, StringComparison.
            ↳InvariantCultureIgnoreCase) >= 0)
            {
                _logger.LogInformation("The browser {currentBrowser} is satisfied_
                ↳using {allowedBrowsers} configuration.");

                return true;
            }
        }

        _logger.LogInformation("The browser {currentBrowser} is not allowed using current_
        ↳toggle configuration.");

        return false;
    }

private string GetCurrentBrowser()
{
    return _contextAccessor.HttpContext
        .Request
        .Headers[UserAgent]
        .FirstOrDefault() ?? string.Empty;
}

```

Finally, we can register our custom toggle using the method `RegisterTogglesFromAssemblyContaining` in our `Startup` class:

```
services.AddEsquio(setup => setup.RegisterTogglesFromAssemblyContaining<Startup>())
```

As you can see, Esquio provides a flexible way to customize as you need.

You can see this sample completed and much more in this [repository](#) and of course, any PR is welcome.

CHAPTER 10

Esquio CLI

Esquio.CLI is a simple crossplatform *dotnet tool* that allow you to interact with *Esquio UI* from the command line. This tool can be used from build or release pipelines on Azure DevOps, Github Actions etc.

To install **Esquio.CLI** open a console window and type the following command using the *.NET Core CLI*:

```
dotnet tools install -g Esquio.Cli
```

Usage: dotnet-esquio [options] [command]

Options: -?|-h|--help Show help information

Commands:

features *Manage Esquio features using Esquio UI HTTP API*

parameters *Manage Esquio parameters using Esquio UI HTTP API*

products *Manage Esquio products using Esquio UI HTTP API*

toggles *Manage Esquio toggles using Esquio UI HTTP API*

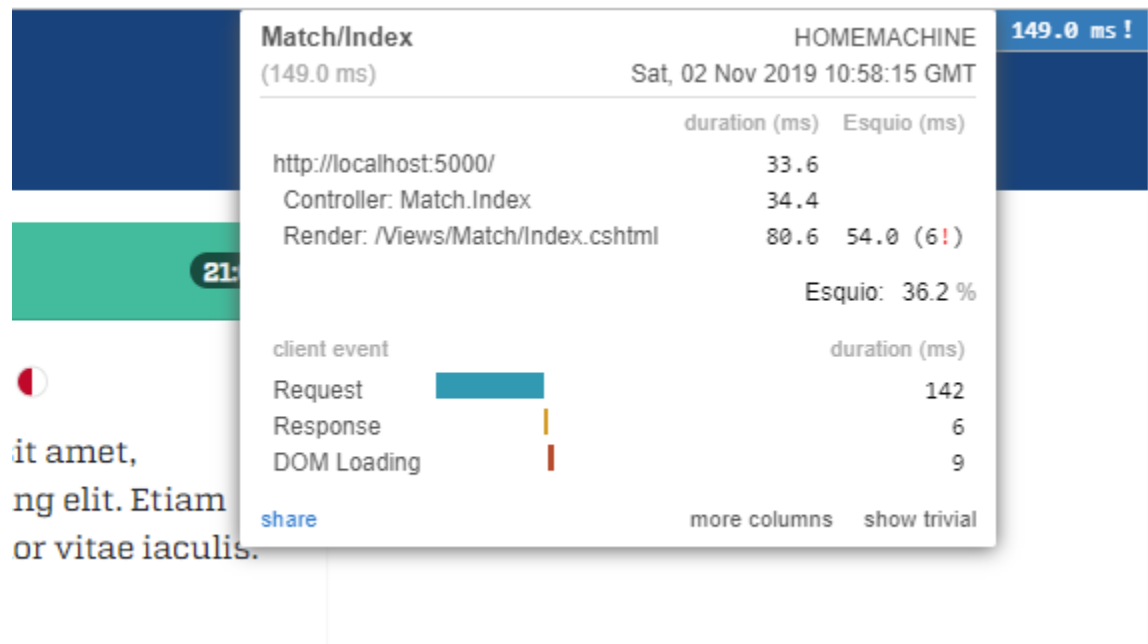
Run 'dotnet-esquio [command] --help' for more information about a command.

CHAPTER 11

Esquio & MiniProfiler

MiniProfiler is a *simple but effective mini-profiler* for .NET. It provides a *Step instrumentation* that you can add to the code you want to explicitly profile. Out of the box, *MiniProfiler* contains plugins for profiling ASP.NET Core, EF Core. But is extensible!.

We use this extensibility to perform profiling over *Esquio* feature evaluations and show the elapsed evaluation time for each feature, each toggle and also show common errors for duplicate feature evaluation or exceptions inside toggle executions.



Call Type Step Duration (from start)	Call Stack Command
61.80 ms	Controller: Match.Index — 34.40 ms
Esquio - Feature Evaluation Render: /Views/Match/Index.cshtml 17.5 ms (T+61.8 ms)	ProcessAsync > Start > Start > MoveNext > IsEnabledAsync > Start > Start > MoveNext > BeginFeatureEvaluation > Write > OnNext Esquio Feature Evaluation default:MatchScore
Esquio - Feature Evaluation ! Render: /Views/Match/Index.cshtml 30.5 ms (T+99.3 ms)	ProcessAsync > Start > Start > MoveNext > IsEnabledAsync > Start > Start > MoveNext > BeginFeatureEvaluation > Write > OnNext Esquio Feature Evaluation default:MatchScore
19.20 ms	Render: /Views/Match/Index.cshtml — 19.20 ms
Esquio - Feature Evaluation Render: /Views/Match/Index.cshtml 3.8 ms (T+129.2 ms)	ProcessAsync > Start > Start > MoveNext > IsEnabledAsync > Start > Start > MoveNext > BeginFeatureEvaluation > Write > OnNext Esquio Feature Evaluation default:DarkMode
Esquio - Toggle Execution Render: /Views/Match/Index.cshtml 1.2 ms (T+131.9 ms)	ProcessAsync > Start > Start > MoveNext > IsEnabledAsync > Start > Start > MoveNext > BeginToggleEvaluation > Write > OnNext Esquio Toggle Execution default:DarkMode:Esquio.Toggles.GradualRolloutUserNameToggle

11.1 Setup

To install **MiniProfiler.Esquio** open a console window and type the following command using the *.NET Core CLI*:

```
dotnet package add MiniProfiler.Esquio
```

or using *Powershell* or *Package Manager*:

```
Install-Package Esquio.AspNetCore
```

To configure MiniProfiler.Esquio on ASP.NET Core, add MiniProfiler service configuration and new *Esquio* plugin with **AddEsquio**:

```
services
    .AddMiniProfiler(options =>
    {
        options.RouteBasePath = "/profiler";
        options.EnableServerTimingHeader = true;

        options.ResultsAuthorize = (_) => true;
        options.ShouldProfile = _ => true;

        options.IgnoredPaths.Add("/lib");
        options.IgnoredPaths.Add("/css");
        options.IgnoredPaths.Add("/js");
        options.IgnoredPaths.Add("/assets");

    }).AddEsquio();
```