Liquid Content Synchronizer

A cross platform application to update your visualizers’ files

Table of Contents

[Introduction 2](#_Toc482291915)

[How to use this app 2](#_Toc482291916)

[Technologies used 3](#_Toc482291917)

[Prerequisites 3](#_Toc482291918)

[How to develop this app 3](#_Toc482291919)

[Getting familiar with the Liquid Content API 3](#_Toc482291920)

[Getting the API Key 5](#_Toc482291921)

[Creating the .NET Core console application 6](#_Toc482291922)

[Adding NuGet packages 7](#_Toc482291923)

[Project structure 7](#_Toc482291924)

# Introduction

This application is intended to let users edit the visualizers’ files in their favorite code editor. This will allow you download the different files that a visualizer contains (header, footer, template, CSS and JavaScript) with a single command and upload the changes with one simple command as well.

With this application, you can be more fluent while you're changing and testing the styles of your visualizers because, once you save your changes in your editor, you only have to upload the files with a simple command and just refresh your browser to see those changes working in your site.

Besides, the application was created in .NET Core, so that it will be used not only in a Windows environment, but in Mac OSX and Linux as well.

This document will explain the steps you have to follow to build a .NET Core application which uses the Evoq Liquid Content API. In this particular example, we will learn how to use the Liquid Content API to list all the visualizers as well as how to get and save the details of a given visualizers.

# How to use this app

This is a console application written in .NET Core so it can be used in a Windows command line interface or in a Mac OSX or Linux terminal.

It has 3 main usages:

1. Get the list of visualizers:
   * You can get only the user defined visualizers:

|  |
| --- |
| dotnet liquid.dll list |

* + Or all the visualizers (system defined visualizers and user defined visualizers):

|  |
| --- |
| dotnet liquid.dll list all |

1. Download the different files (HTML, CSS and JavaScript) of a single visualizer:

|  |
| --- |
| dotnet liquid.dll download <VisualizerId> |

This command will create a folder with the visualizer ID as its name, and with the HTML, CSS and JavaScript files inside it.

1. Upload the HTML, CSS and JavaScript files of a single visualizer:

|  |
| --- |
| dotnet liquid.dll upload <VisualizerId> |

There must be a folder with the ID of the visualizer and with the HTML, CSS and JavaScript files inside it (same filenames that the download option creates). The following files are updated:

* **footer.html**
* **header.html**
* **scripts.js**
* **styles.css**
* **template.html**

1. Keep your local changes synchronized

|  |
| --- |
| dotnet liquid.dll sync <VisualizerId> |

This option downloads your visualizers’ files (like the download option) and then it monitors that folder, so that if one of the files changes, it will automatically synchronize those changes in your site.

So, you can start by listing all the visualizers with the list parameter to see the ID of the visualizer you want to modify.

Then use the download parameter to download all the files of that visualizer.

After that, you can start modifying the different files you want to modify.

And when you want to see the changes in your site, use the upload command and refresh the page where you have this content item.

Or just use the sync option and you only have to modify those files in your editor, save the changes and reload the page in your site to see those changes in action.

# Technologies used

* Liquid Content API from Evoq
* .NET Core 1.1
* Visual Studio 2017

# Prerequisites

To follow this tutorial, you will need:

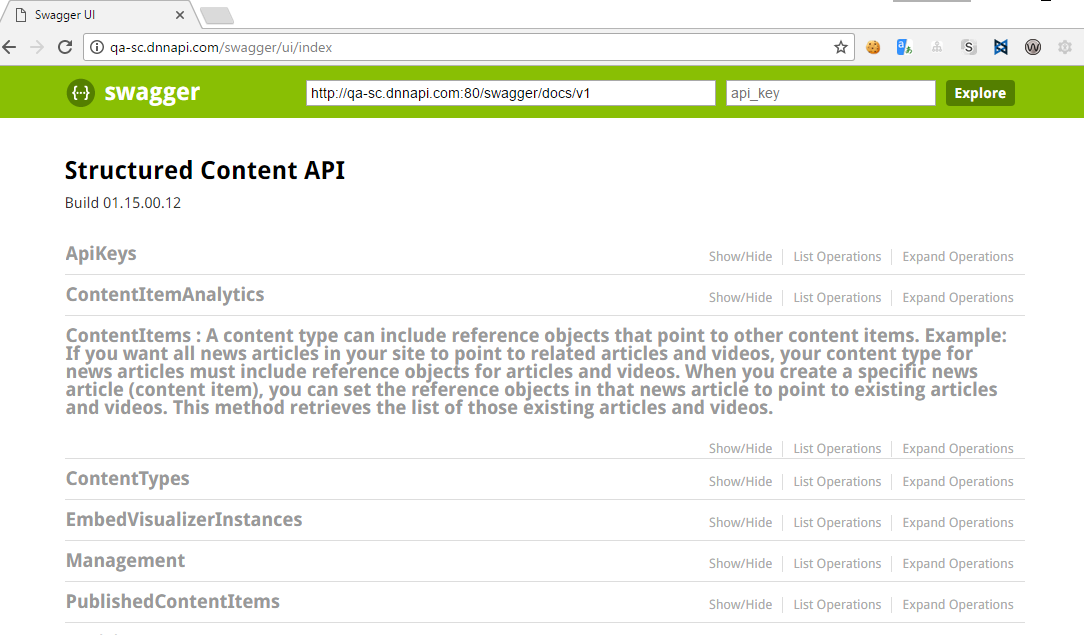
* [Visual Studio 2017 with .NET Core](https://www.microsoft.com/net/core#windowsvs2017)
* [An Evoq Content or Engage](http://www.dnnsoftware.com/products) site with Evoq Liquid Content enabled

# How to develop this app

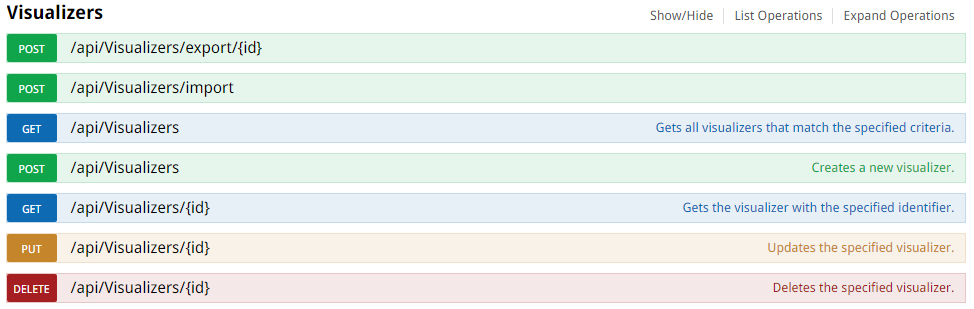
The following steps will guide you throw the process of knowing how are the Liquid Content API methods that we will use to get the list of visualizers, to get a particular visualizer and to update a visualizer. Let’s start by taking a look at the API.

## Getting familiar with the Liquid Content API

Liquid Content API is built following the [OpenAPI Specification](https://github.com/OAI/OpenAPI-Specification) (aka The Swagger Specification). Swagger allows you to see the details of the API through its web interface. In our case, the Liquid Content API can be examined through <http://qa-sc.dnnapi.com/swagger>



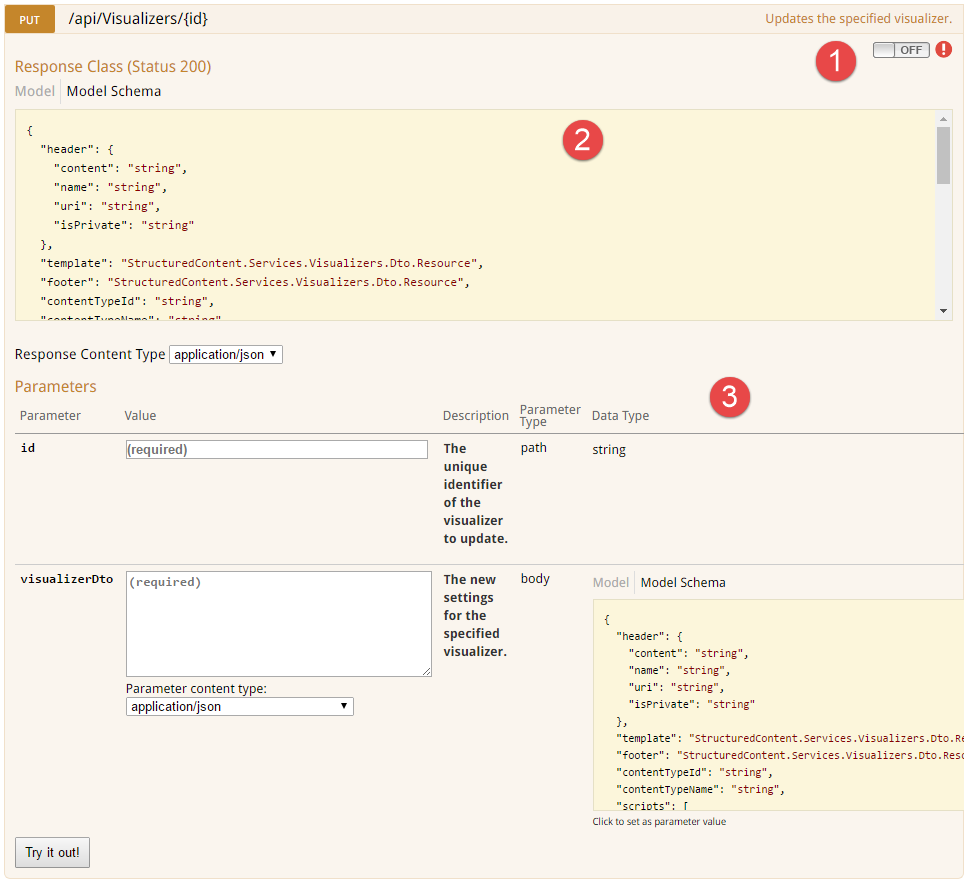
In the Swagger main page, you can see the different sections of the Liquid Content API. This tutorial will use only one of those sections (Visualizers) and if you click on that section, you will see the different API methods allowed to deal with the visualizers:



Our application will use three of those methods:

* [GET /api/visualizers](http://qa-sc.dnnapi.com/swagger/ui/index#!/Visualizers/Visualizers_GetVisualizersByContenttypeidAndSearchtextAndStartindexAndMaxitemsAndFieldorderAndOrderascAndCreatedfromAndCreatedto), to get the list of all visualizers
* [GET /api/Visualizers/{id}](http://qa-sc.dnnapi.com/swagger/ui/index#!/Visualizers/Visualizers_GetVisualizerById), to get the details of a particular visualizer
* [PUT /api/Visualizers/{id}](http://qa-sc.dnnapi.com/swagger/ui/index#!/Visualizers/Visualizers_PutVisualizerByIdAndVisualizerdto), to update the details of a particular visualizer

To see how those method works, in terms of parameters, payload, etc. you only have to click on those methods and the UI will show you what’s needed to call the API method. Let’s take a look at the PUT method, to update a single visualizer:



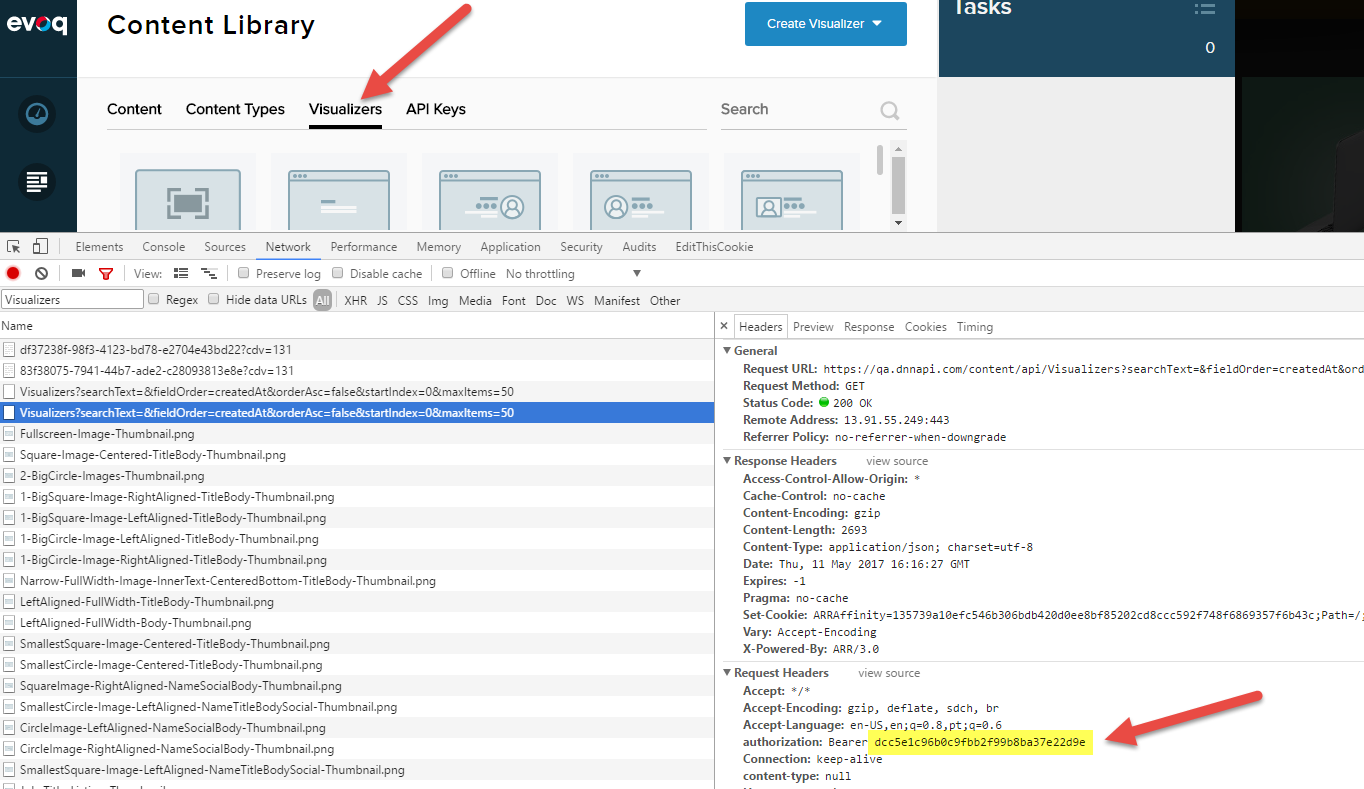
* Credentials: The API uses OAuth, so you have to have a valid OAuth Token in order to use the API (more on this in the following section)
* Response: This is the object that is returned by the PUT method, if the update completes successfully
* Parameters:
  + Id: The visualizer ID
  + visualizerDto: The details of the visualizer that you want to save (name, template, CSS, Scripts...). As you can see in the screenshot, the “Parameter Type” is “body”, which means that this JSON file is the payload of that PUT request

## Getting the API Key

If you want to start using the API, the first thing you need is an API Key. This is basically an OAuth Token that allows you to use the API. Without the API Key, your requests will always end up in a 401 (Unauthorized) response.

The Evoq Content Library page of the Persona Bar lets you generate an API key, but at the moment this tutorial was created, that API key is only to deal with Content Items and Content Types.

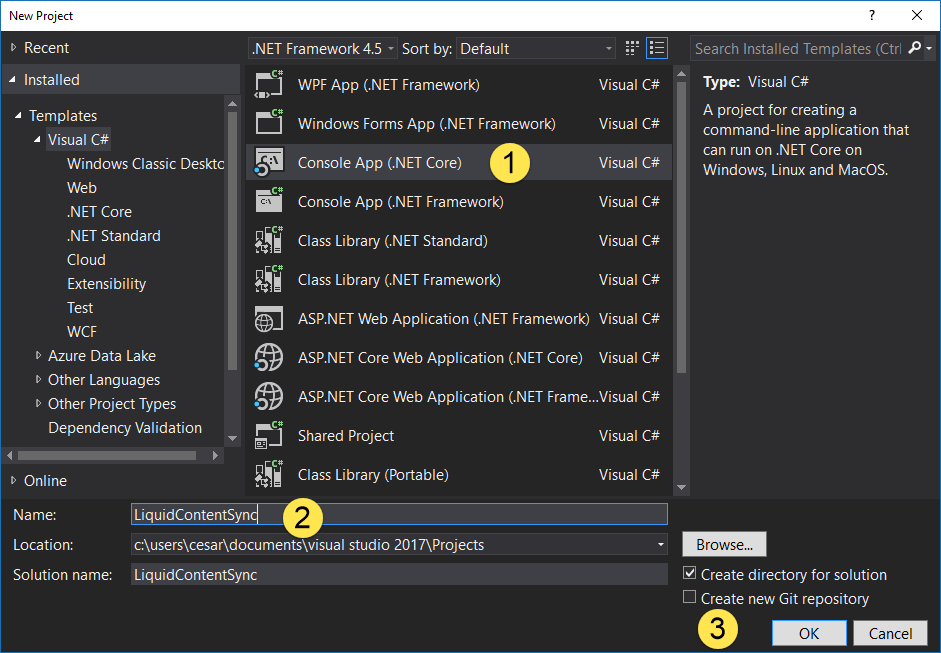
But there’s a way to get a valid API Key, which is by inspecting the web browser and, in the Network tab, look for a request to the Visualizers API (you have to navigate to the visualizers section of the Content Library to see those requests), and get the Authorization header from the requests:



With that token in hand, we can now start calling the API. Let’s start with our app.

## Creating the .NET Core console application

To create a .NET Core application in Visual Studio 2017, go to File > New > Project and from the “New Project” dialog box, select “Console App (.NET Core)”. Give it a name, and click Ok:



This will create a HelloWorld application.

### Adding NuGet packages

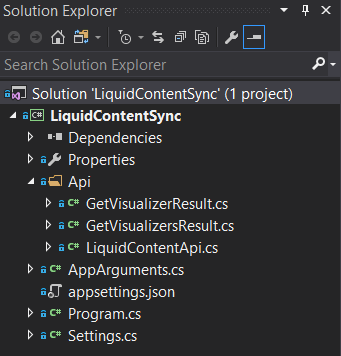
This application uses the following NuGet packages:

* **Microsoft.NETCore.App**: This is installed by default by VS2017
* **System.Runtime.Serialization.Json**: This is needed to serialize our model to JSON object and vice versa when chatting with the API
* **Microsoft.Extensions.Configuration** and **Microsoft.Extensions.Configuration.Json**: To make easy the access to the application settings, which is a JSON file where we will store our API URL and API Key

So, right click on your project name in the Solution Explorer, select Manage NuGet Packages and install the above packages.

### Project structure

Let’s have a look at the project structure:



Under the API folder, we have the following files:

* **GetVisualizerResult.cs**: This is the JSON object we can see in Swagger for the GET Visualizer operation (you can copy that JSON from Swagger and in Visual Studio, paste it as a C# object through Edit > Paste Special > Paste JSON As Classes
* **GetVisualizersResult.cs**: JSON object returned from the GET Visualizers API method
* **LiquidContentApi.cs**: This is the class we’re going to use to call the API (GETs and PUT method)

The file **AppArguments.cs** is a helper to deal with the application parameters.

The file **Settings.cs** is just a helper class to read the settings from the application settings file (**appsettings.json**), which is a JSON file with the two settings used by the app (API\_URL and API\_key).

And finally, the file **Program.cs** is where we read the application parameters, and call the 3 different API methods to get the list of visualizers, to get a particular visualizer or to update a visualizer.