DNN LC File Debug

# Introduction

The purpose of "DNN LC File Debug" is to display the files associated with a visualizer from a browser. When we define Script and Style in a visualizer, in the rendering of the page two files are downloaded: .js and .css. These files are downloaded from an unfriendly path that can generate confusion when debugging problems on the client side.

A Chrome extension has been created that displays a live log of the .js and .css files of the visualizer that loads the page.

# Technologies Used

* Liquid Content APIs from Evoq
* Chrome Extension
* JavaScript

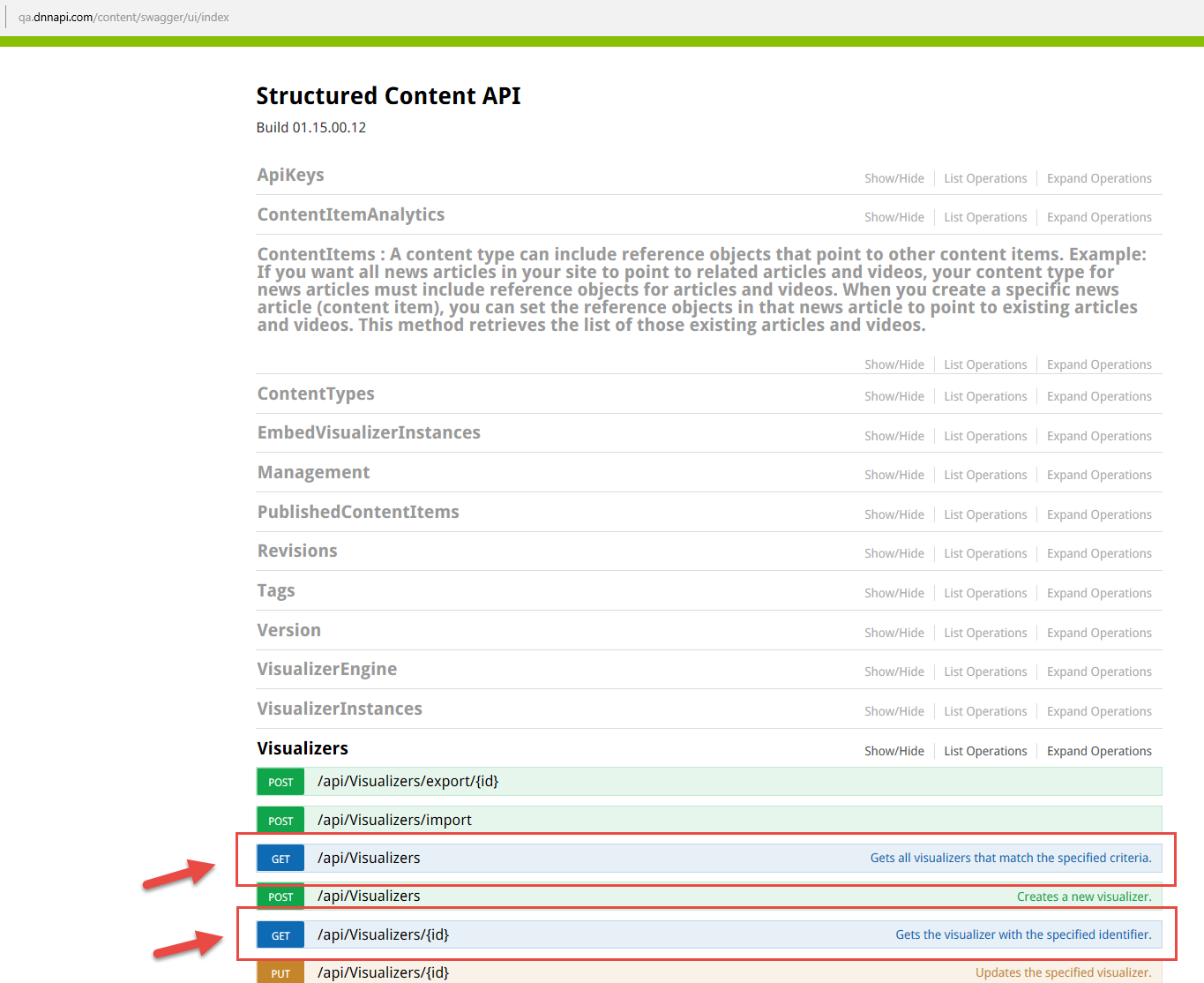
# Prerequisites

* DNN Evoq site updated to the latest version
* Chrome browser updated to the latest version
* JavaScript Text editor

# How To

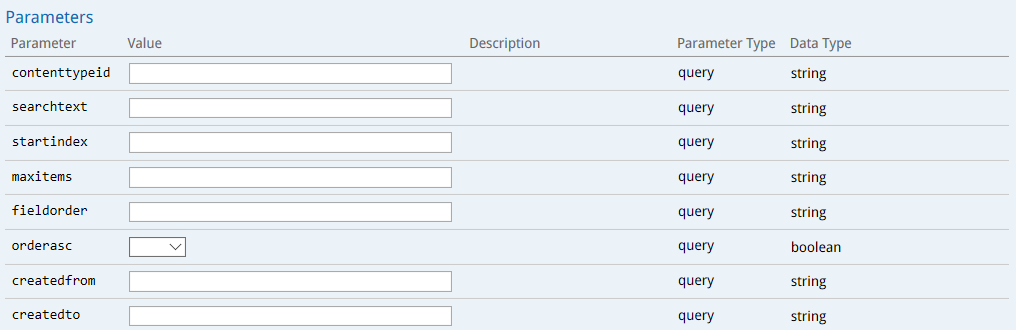
## Access Liquid Content APIs

Visit **Swagger**page to get a list of API methods for Liquid Content in <https://qa.dnnapi.com/content/swagger/ui/index> .

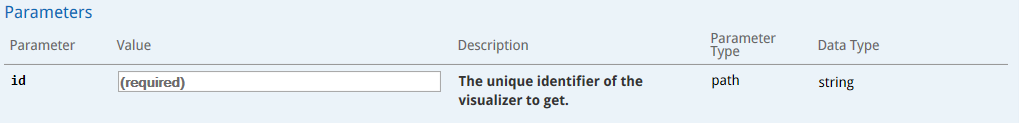


Click on List Operations to expand each group and see available methods. For this application, we will use the methods of the group **Visualizers.** We will use two methods of the API to obtain the list of visualizers and another method to obtain a visualizer by identifier.

* **[GET] /api/Visualizers:** Gets all visualizers that match the specified criteria.

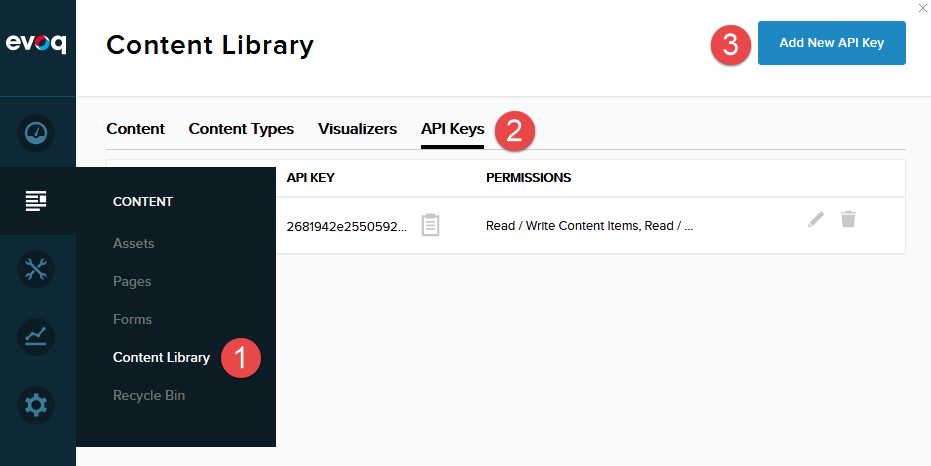


* **[GET] /api/Visualizers/{id}:** Gets the visualizer with the specified identifier.

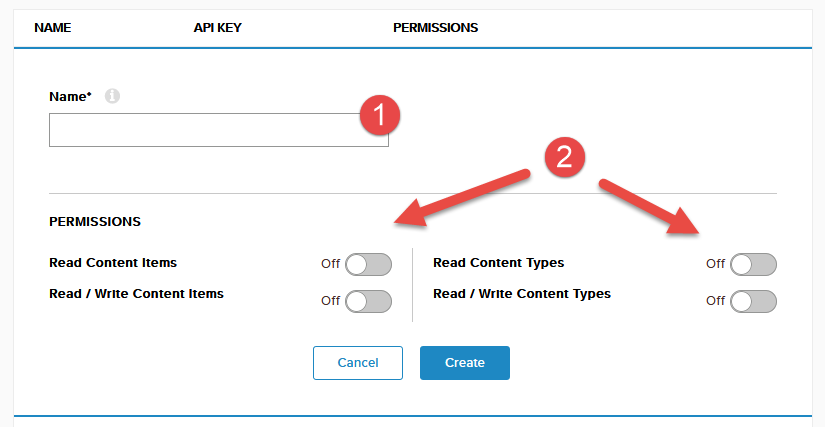


## Accessing the API with API key

Liquid content provides access to the APIs through API keys. This process can be done in "*Content Library*", "*API Keys*" tab and clicking on "*Add New API Key*".

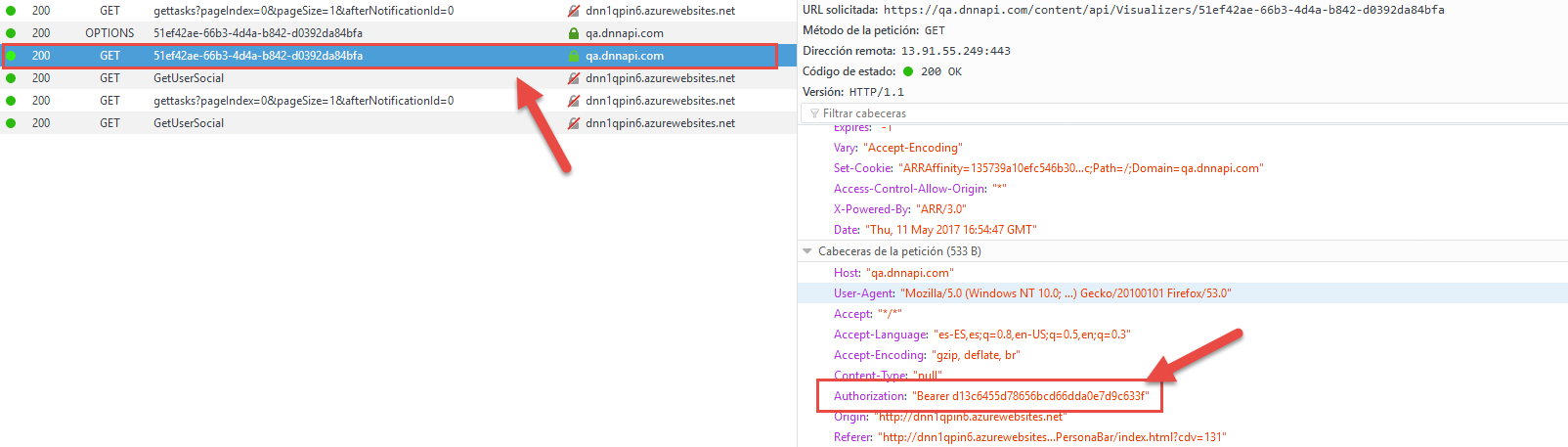


At this time we must provide a name and permissions, to finish the process and obtain a valid API Key.



As we can see in the previous image, in the latest version of Liquid Content it is not possible to obtain a valid API Key for access to the viewers. For access to the methods of the visualizers and until we have a new version of Liquid Content, we will get the API Key through **the network tab of your browser developer tools**.

Perform editing operations of some of your visualizers on your DNN site, check the requests made and in the header get the value of the Key API in the "*Authorization*" field. You must perform this operation every time the API Key expires.



## Building the Chrome Extension

Extensions allow you to add functionality to Chrome without diving deeply into native code. You can create new extensions for Chrome with those core technologies that you're already familiar with from web development: HTML, CSS, and JavaScript.

### The manifest

We'll need to create is a manifest file named manifest.json. This manifest is a metadata file in JSON format that contains properties like your extension's name, description, version number and so on. We will use it to declare to Chrome what the extension is going to do, and what permissions it requires in order to do those things.

|  |
| --- |
| {  "name": "DNN LC File Debug",  "description": "Displays the live log with downloaded files associated with a visualizer (.js and .css)",  "version": "0.1",  "permissions": [  "debugger"  ],  "background": {  "scripts": ["background.js"]  },  "browser\_action": {  "default\_icon": "icon.png",  "default\_title": "DNN LC File Debug"  },  "manifest\_version": 2  } |

### Permissions

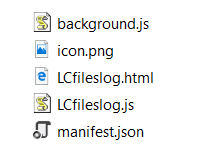
The *chrome.debugger* API serves as an alternate transport for Chrome's remote debugging protocol. We use *chrome.debugger* to attach to one or more tabs to instrument network interaction, debug JavaScript, mutate the DOM and CSS, etc.

### Resources

You probably noticed that manifest.json pointed at two resource files when defining the browser action: **icon.png** and **background.js**. Both resources must exist inside the extension package.

* *Icon.png* will be displayed next to the Omnibox, waiting for user interaction.
* *background.js* meets a common need for extensions to have a single long-running script to manage some task or state.

Extension content:



We have files that do not appear in the manifest file:

* *LCfileslog.html* html code page shows the log
* *LCfileslog.js* javascript code with methods to display the log

### The code

The main page html schema is in the "*LCfileslog.html*" file. Here is added the reference to the javascript file that contains the main code.

|  |
| --- |
| <html>  <head>  .  .  .  <script src="LCfileslog.js"></script>  </head>  <body>  <div id="container"></div>  </body>  </html> |

The file "*background.js*" initializes the debugger and creates the window where all log entries will be displayed.

|  |
| --- |
| chrome.browserAction.onClicked.addListener(function(tab) {  chrome.debugger.attach({tabId:tab.id}, version,  onAttach.bind(null, tab.id));  });  var version = "1.0";  function onAttach(tabId) {  if (chrome.runtime.lastError) {  alert(chrome.runtime.lastError.message);  return;  }  chrome.windows.create(  {url: "LCfileslog.html?" + tabId, type: "popup", width: 800, height: 600});  } |

The file "*LCfileslog.js*" contains all the code necessary to display the log entries. In addition to the code to obtain the data of the visualizers through the Liquid Content API.

Call to the Liquid Content API to get the visualizers list:

|  |
| --- |
| function getVisualizerData(fileUri) {  var xhttp = new XMLHttpRequest();  xhttp.open("GET", "https://qa.dnnapi.com/content/api/Visualizers?maxitems=50", false);  xhttp.setRequestHeader("Authorization", "Bearer " + apiKey);  xhttp.setRequestHeader("Content-type", "application/json");  xhttp.send();  var visualizers = JSON.parse(xhttp.responseText);  .  .  . |

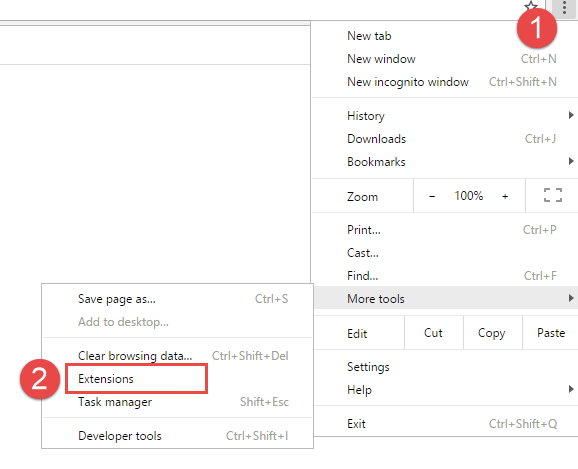
Check that the visualizer has a file whose path matches that of the downloaded file (.js or .css):

|  |
| --- |
| .  .  .  xhttp = new XMLHttpRequest();  xhttp.open("GET", "https://qa.dnnapi.com/content/api/Visualizers/" + visualizers.documents[i].id, false);  xhttp.setRequestHeader("Authorization", "Bearer " + apiKey);  xhttp.setRequestHeader("Content-type", "application/json");  xhttp.send();  var visualizer = JSON.parse(xhttp.responseText);    // Check that the path of the file matches with the .css file for the visualizer  if (visualizer.cssFiles.length > 0) {  if (fileUri.includes(visualizer.cssFiles[0].uri)) {  return [visualizers.documents[i].id, visualizers.documents[i].name, 'css'];  }  }    // Check that the path of the file matches with the .js file for the visualizer  if (visualizer.scripts.length > 0) {  if (fileUri.includes(visualizer.scripts[0].uri)) {  return [visualizers.documents[i].id, visualizers.documents[i].name, 'script'];  }  }  .  .  . |

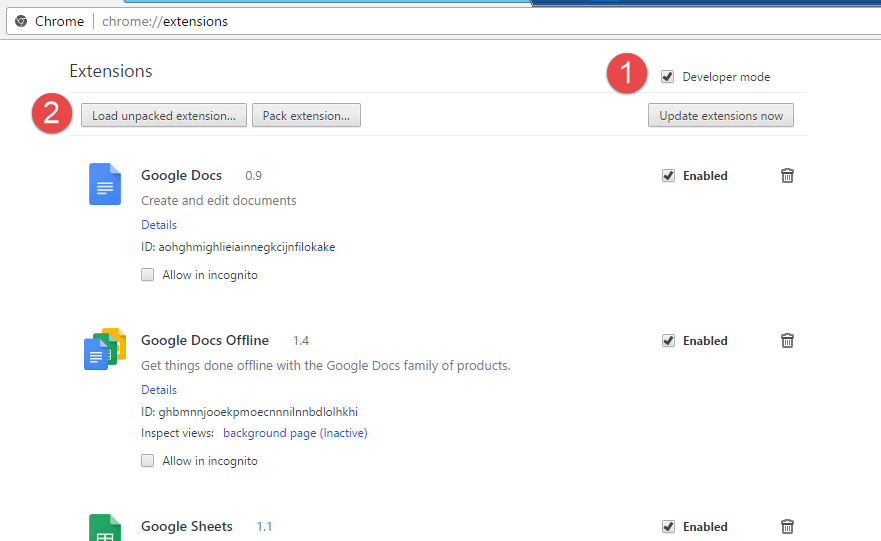
### Load the extension

Chrome gives you a quick way of loading up your working directory for testing.

1. Visit chrome://extensions in your browser (or open up the Chrome menu by clicking the icon to the far right of the Omnibox: The menu's icon is three horizontal bars. and select Extensions under the Tools menu to get to the same place).



1. Ensure that the Developer mode checkbox in the top right-hand corner is checked.
2. Click Load unpacked extension… to pop up a file-selection dialog.

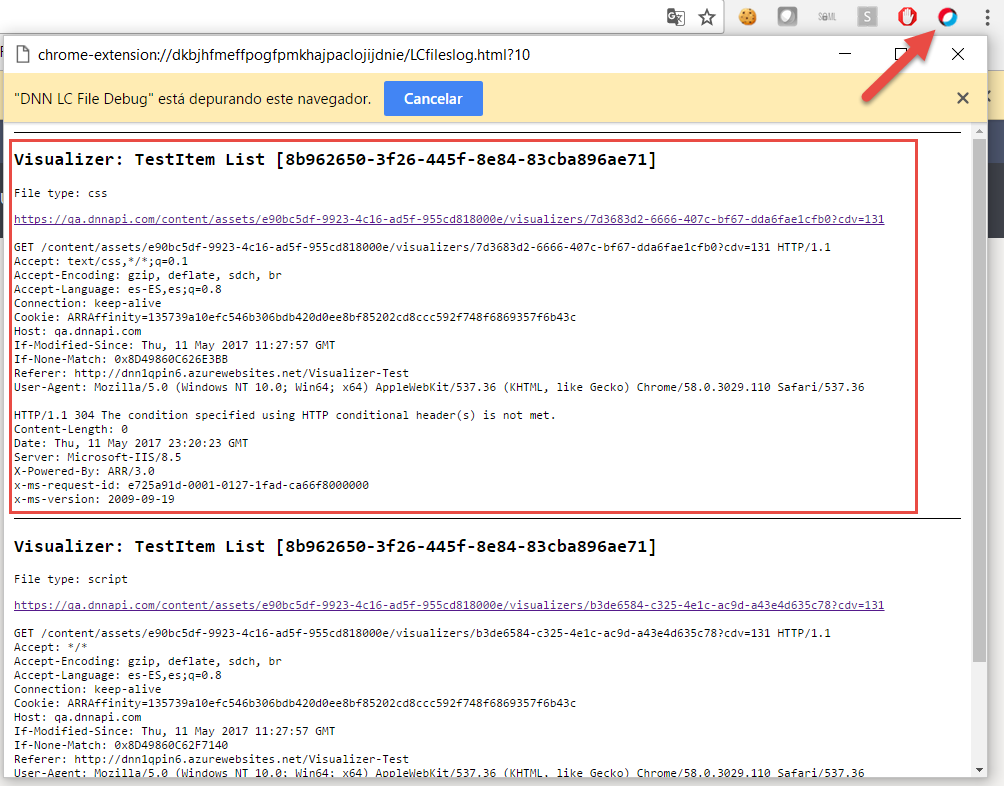


1. Navigate to the directory in which your extension files live, and select it.

## Working with the extension

Once the extension is installed we can use it. If we click on the icon a new window is displayed. This new window shows all the entries of the files associated to the visualizer.

The extension displays in the log the .js and .css files associated with the visualizer. It also shows a link for downloading the file and all the data of the request and response.



# Conclusion

In this demo we tried to give extra information about a page of a DNN site that uses visualizers. We made a simple connection to the Liquid Content API using Javascript, language that uses a Chrome extension. As a final result we have a live log of the files associated to a visualizer, direct access to the code of these files and information of the visualizers used in the page.

In later developments we could launch the debugging of the javascripts files from the log.