

SiteKiosk API Setup

How and where to perform scripting with the SiteKiosk API

You have multiple options to use scripting to enhance your own or remote pages, in this document the multiple ways to do this will be explained.

The Advanced Config

The Advanced Config is only available when using the expert mode of the Content Editor.

Any JavaScript code you want to put as value into the **Advanced Config** needs to be encoded, as pure JavaScript will break the system.


Following online tool will help you encode such scripts:

https://coderstoolbox.net/string/#!encoding=js&action=encode&charset=utf_8


Expert Mode

To activate the expert mode of the Content Editor you need to append **&expert** to the URL of the project you are displaying in the WebBrowser.

For example you have opened a project and this is being displayed in the WebBrowser:

 sitekiosk.online/projects/editor/assets/index.html#app-editor?projectId=637f347d6e9ff62c774cd4cf

You need to edit the url, so that it will look like this (look at the end of the URL):

 <https://sitekiosk.online/projects/editor/assets/index.html#app-editor?projectId=637f347d6e9ff62c774cd4cf&expert>

Now load this URL **and** reload the page after it has been loaded. You should now see the following elements at the toolbar of the content editor:



With this you have activated the expert mode which allows you to use the **Advanced Config** and gives you the ability to add content scripts, which are described later in this document.

SiteKiosk API

SiteKiosk API

The SiteKiosk API is the API with which you can use SiteKiosk methods in either the Android or the Windows client. The SiteKiosk API is a system which encapsulates both SiteKiosk Online versions (Windows and Android).

It provides the ability to access the system, its devices, the current displayed content and state.

To know the abilities of the SiteKiosk API you can have a look into the SiteKiosk API documentation, which you can find here: <https://sitekiosk.online/projects/external/apidocs/generated/index.html>

The SiteKiosk API always begins with **siteKiosk**. followed by the modules, methods or properties you want to use. For example, to write a log message your code will look like this:

```
siteKiosk.log.info("Hello World");
```

Internal SiteKiosk API

As we only move methods into the SiteKiosk API when there is a customer demand for it, the scope can be a little bit limited for more specialized needs. For this there is an internal SiteKiosk API, which is different for each of the SiteKiosk Online client versions (Android and Windows). The SiteKiosk API will wrap the internal SiteKiosk API.

We won't provide in depth documentation for it, but be assured that most wishes can be fulfilled. Just ask us and we will find or create the solution for it.

The internal SiteKiosk API always begins with **__siteKiosk**. followed by the modules, methods or properties you want to use. For example, to write a log message your code will look like this:

```
__siteKiosk.log.info("Hello World");
```

SiteKiosk API in the global SiteKiosk context

To run a script inside the global SiteKiosk context after SiteKiosk has been loaded, without any dependency to other pages or the content player, you can add any JavaScript code, encoded into the config path **system.startupScript**. Just add a new entry into the **Advanced Config** using this path and your encoded JavaScript code.

The provided script will be run as an async function.

SiteKiosk API inside the Content Player

To write scripts which should run inside the content player you first need to activate the expert mode of the Content Editor. When this is done you can proceed.

Now you need to open the **Settings->Expert** dialog.

Settings Help (Expert)

Project

- General & User rights
- Zoom
- Fonts
- RSS
- Languages
- Variations
- Variables
- Expert**

Client

- User Interface
- Browser
- Content Filter
- Downloads
- Idle & Logout
- Printing
- System
- Security
- Smart Kiosk Control
- Advanced

Scripts

Files

Upload additional files you want to use in your content. Script and style files are inserted into the page when it is loaded. Other files are pushed to the clients and can then be referenced within your custom scripts and styles.
The script files uploaded here are initiated when the player is started

Upload script files

Script files	

Select file

Upload style file (CSS)

These style files are also loaded into the editor and can therefore cause side effects

Style files	

Save Cancel

Inside the settings dialog add a new script by pressing the "+ Add" button. You can e.g. then enter your JavaScript code which should be executed.:

The screenshot shows the 'Settings' dialog with the 'Expert' tab selected. The 'Scripts' section is active, showing a text input for 'Name' with the value 'New_Script' and a trash icon. Below it is a code editor with the text '1 console.log("New Script");'. A '+ Add' button is located below the code editor. The 'Files' section contains an informational message: 'Upload additional files you want to use in your content. Script and style files are inserted into the page when it is loaded. Other files are pushed to the clients and can then be referenced within your custom scripts and styles. The script files uploaded here are initiated when the player is started'. Below this is an 'Upload script files' section with a table for script files. At the bottom right of the dialog are 'Save' and 'Cancel' buttons.

Script files	

Such script will be executed after each logout and start of SiteKiosk Online. You need to remember this, because some actions are permanent (like registering devices).

SiteKiosk API inside a WebPage you control

If you have full control over a WebPage you can grant SiteKiosk API access to your page. You do this by writing an URL template which matches your page URL

For example, if you want to grant the URL `https://www.somedomain.com/myPage.html` the needed rights you can use exactly this URL.





If you want to provide the rights to all pages with `https://www.somedomain.com/subDirectory` in its URL you need to use `https://www.somedomain.com/subDirectory/*` as URL template.

To add a URL template for the SiteKiosk API rights open **Settings->Client** and scroll to the **URLS with SiteKiosk Object Model Permission** sub section. Then add the template, after you finished this, it should look like this:

URLs with SiteKiosk Object Model Permission Windows Android

For security reasons, the SiteKiosk Object Model will only work in the paths/URLs defined here.

[Add URL](#)

URL / Path	Action
<code>https://www.somedomain.com/myPage.html</code>	 
<code>https://www.somedomain.com/subDirectory/*</code>	 

Now your page can use the SiteKiosk API when it is loaded inside a WebPage element. Simply use it in a `<script>` DOM element or use it in a script you have loaded by a `<link>` DOM element. You can directly use the SiteKiosk API, you don't need to initialize anything and you don't need to import additional modules.

SiteKiosk API inside a WebPage you **don't** control

This is a more complex case, but the SiteKiosk Online **Windows** Client provides the ability to do it.

We provide two solutions for it.

Solution 1

The first solution is by injecting a script into every page before it is loaded, we call those preload scripts. To use these you need to enhance the SiteKiosk Online configuration and an array of preload script rules.

You do this by creating a new Advanced Config element in **Settings->Client** and scroll to the **Advanced Config** sub section and then add a new entry for the path ***system.browser.preloadScripts***.

Now you can put JSON content which describes for which URL template the preload script should be applied and where to find the script file. The following sample should be self describing, but you can always look up the needed fields [HERE](#).

```
[
  {
    "filter": {
      "url": "https://www.somedomain.com/myPage.html"
    },
    "script": "C:\\\\SomePath\\myScript.js"
  }
]
```

The clear downside of the approach is that you need to have the preload script file on the client machine.

Solution 2 (Recommended)

The second solution is by using the SiteKiosk Online web page automation system. You specify the URL template for which the automation should be applied and provide an array of steps which should be executed. One of these steps is the **ExecuteScript** step. The JavaScript code is encoded inside the step. To encode or decode JavaScript you can use this online tool: http://coderstoolbox.net/string/#!/encoding=js&action=encode&charset=utf_8

You can add web page automation by using the **Advanced Config**, like in the first solution, but in this solution you need to use the ***system.browser.webPageAutomation*** config path.

The following sample uses this script to be run when the URL <https://www.somedomain.com/someSubPage> has been loaded:

```
1 return new Promise(resolve => {
2   const smartCardReaderSensor = __siteKiosk.io.devices.findByName("SmartCardReader");
3   smartCardReaderSensor.on(smartCardReaderSensor.events.dataGenerated, onSensorDataGenerated);
4   onSensorDataGenerated(smartCardReaderSensor, smartCardReaderSensor.lastData);
5
6   function onSensorDataGenerated(sensor, sensorData) {
7     if (!sensorData || sensorData.isCardRemoved || !sensorData.smartCard.name) return;
8
9     smartCardReaderSensor.off(smartCardReaderSensor.events.dataGenerated, onSensorDataGenerated);
10
11     setTimeout(resolve, 2000);
12   }
13 });
```

This script will wait until a readable smart card has been inserted and will then wait 2 seconds before the script finishes. You can either use Promises or just perform the code without it, this is up to you. The step will be marked as finished when the Promises resolves and will continue executing any other steps.

This is the JSON content for the web page automation which can be put into the value of the config path:

```
[
  {
    "name": "WaitUntilUserInsertsSmartCard",
    "blockPage": false,
    "filter": {
      "url": "https://www.somedomain.com/someSubPage"
    },
    "steps": [
      {
        "expectedStepResult": true,
        "ignoreStepResult": false,
        "executeScript": {
          "script": "return new Promise(resolve => {\n\tconst smartCardReaderSensor\n= __siteKiosk.io.devices.findByName(\"SmartCardReader\");\n\tsmartCardReaderSensor.on(smartCardReaderSensor.events.dataGenerated,\nonSensorDataGenerated);\n\ttonSensorDataGenerated(smartCardReaderSensor,\nsmartCardReaderSensor.lastData);\n\n\tfunction onSensorDataGenerated(sensor,\nsensorData) {\n\t\ttif (!sensorData || sensorData.isCardRemoved || !\nsensorData.smartCard.name) return;\n\n\t\tsmartCardReaderSensor.off(smartCardReaderSensor.events.dataGenerated,\nonSensorDataGenerated);\n\t\t\t\n\t\tsetTimeout(resolve, 2000);\n\t\t}\n});",
          "timeoutInMs": 0
        }
      }
    ]
  }
]
```

Here a much simpler sample, which you can use as a template, this sample will only log a message into the SiteKiosk Online log:

```
[
  {
    "name": "LogHelloWorld",
    "blockPage": false,
    "filter": {
      "url": "https://www.somedomain.com/someSubPage"
    },
    "steps": [
      {
        "expectedStepResult": true,
        "ignoreStepResult": false,
        "executeScript": {
          "script": "siteKiosk.log.info('Hello World!')",
          "timeoutInMs": 0
        }
      }
    ]
  }
]
```

The web page automation feature is also a great way to remove and/or replace the content of a whole page, automatically click elements, eg.