Templating

Information on scripts and templates. Adhese does not restrict advertising to the use of IAB Standard Formats only. The Adhese Templates solution allows you to wrap more complex creatives such as overlayers, takeovers or floor ads by adding advanced functionality using JavaScript, HTML and CSS.

Advar Templates

Adhese has introduced a new template format called *Advar*. Advar templates are pre-defined creatives to create sophisticated ads consisting of Javascript, CSS, custom JSON objects etc. The results of Advar templates are pre-made ads, such as a text ad including a small image.

Advar templates can have multiple components (images, videos, texts and URLs) that form the final creative. The user provides these elements (or parameters) when they create a new Advar ad. The user responsible for adding a new creative to a campaign does not require coding knowledge in HTML, CSS or JavaScript. The template predefines the code and presents it as a form to the user.

See Add an Advar creative.

An Advar template always consists of two files:

1. The **creative's actual content** contains parameters that refer to the properties of an uploaded creative and its files. Besides any custom parameter defined by the creator of the Advar template, the creative content can also contain several fixed parameters.
2. A **descriptive file** generates the form that the user will see and use when uploading an Advar creative. The file contains a JSON object that defines the different elements available in the template. It will determine how to render the form to the user.

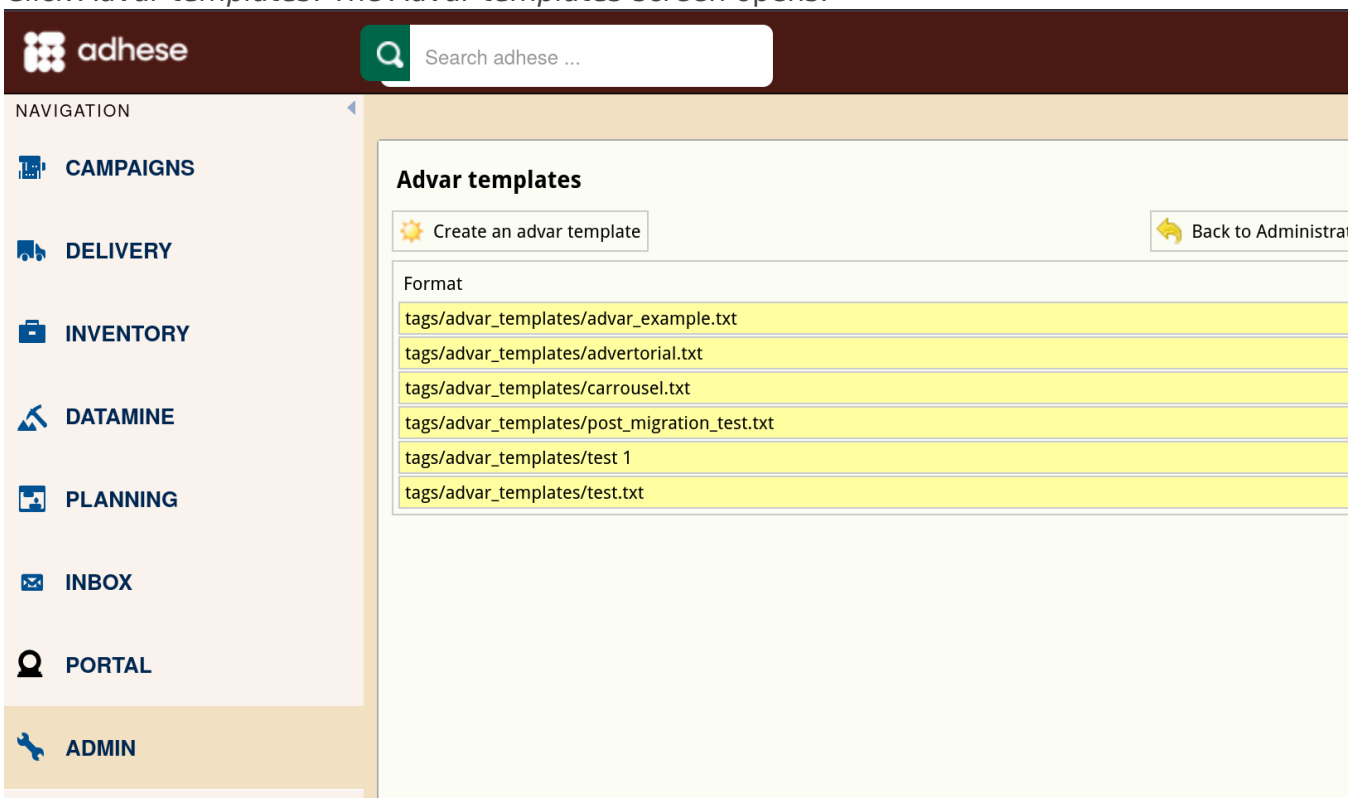
As with templates, an Advar template can also contain a request element as a parameter. An Advar template can use all request properties as content. For example, the creative can display the visitor's name if the call to the ad server contains this parameter (and if the value of the name is available, of course).

The *Advar templates* screen lists the name of the Advar template in the *Format column*.

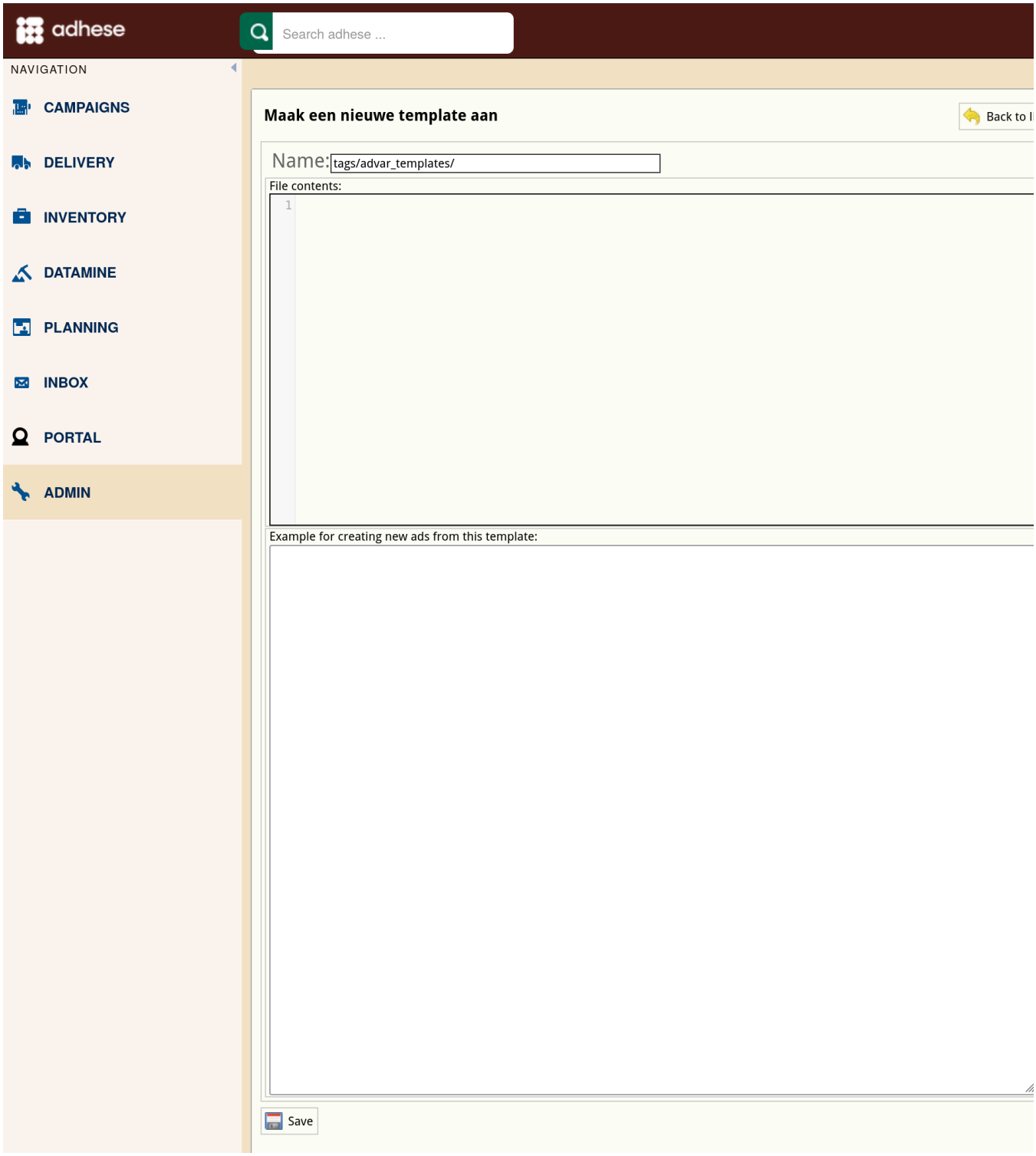
Create a new Advar template

To create a new Advar template:

1. Go to the *Administration* screen. Click *Admin* in the left navigation menu.
2. Click *Advar templates*. The *Advar templates* screen opens:



3. Click *Create an Advar template*. The *Create a new Advar template* screen opens:



4. Enter a name in the **Name** field. Choose a clear and logical name, such as *pp-textad.txt*.
5. Insert the code in the **File contents** field. This code represents the actual content of the creative and includes parameters that refer to the properties of the uploaded creative and its files. Refer to the Appendix Parameters for templates and Advar templates for an overview of the available parameters. Here is an example of some pieces of code:

```
<style>
div {background-image:url(<ADHESE_SWF_SRC_3RD>)}
</style>
<div>
<h1><advar_title></h1>
<p>
<advar_text>
<img src='<ADHESE_SWF_SRC_2ND>' width='<ADHESE_WIDTH>' height='<ADHESE_HEIGHT>'/>
</p>
<p>
<advar_source>
</p>
</div>
```

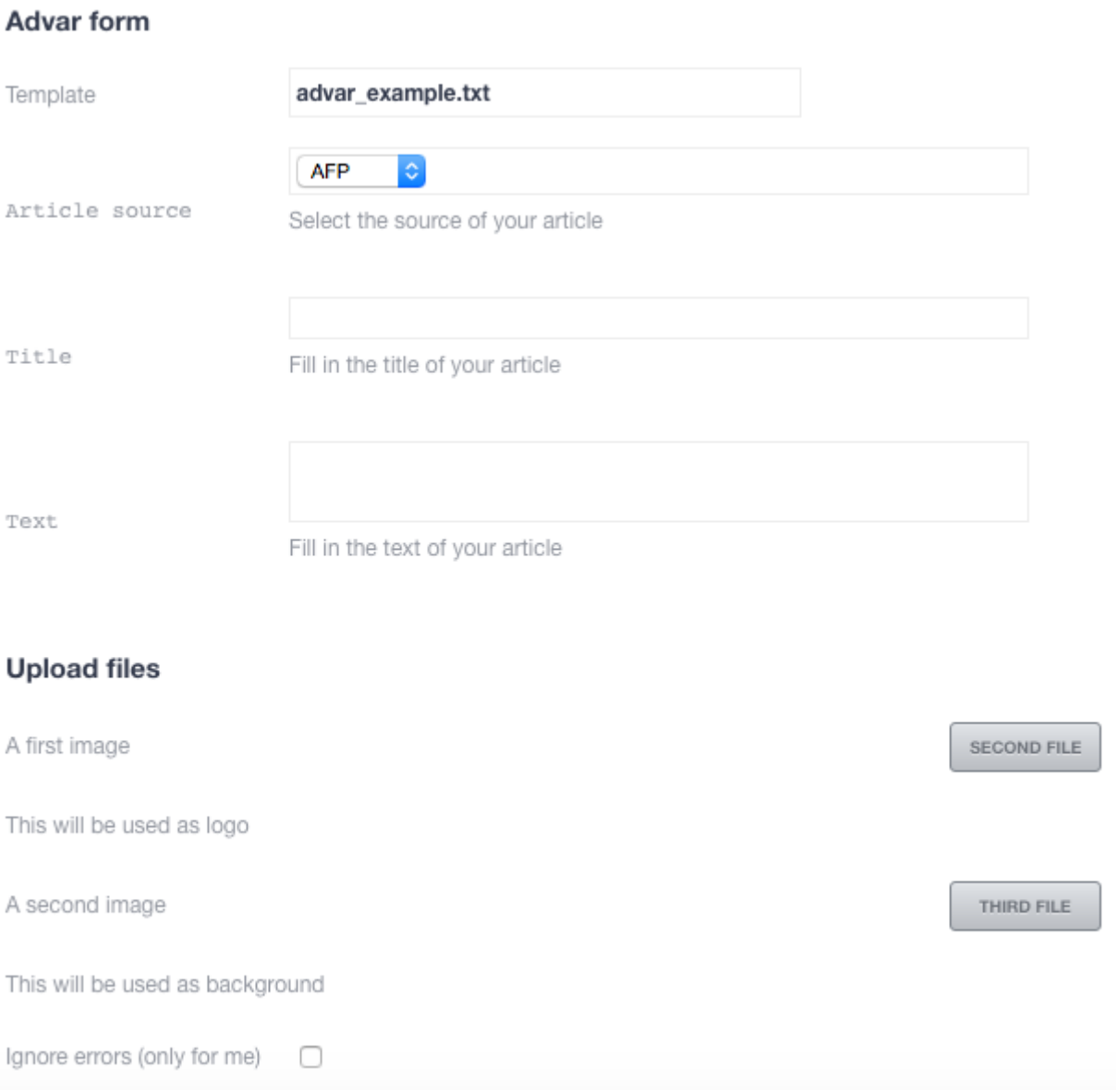
Add inline style attributes to an element to specify the design of the ad.

6. The **Example for creating new ads from this template** field contains the **descriptive file** and makes it possible to create input fields when using an Advar template file to upload an ad. These input fields become visible when you select an Advar template as a creative. The descriptive file contains a JSON object that defines the different elements available in the template. It will determine how to render the form to the user. It can contain three types of fields:
   1. singleLineText,
   2. multilineText, and
   3. select (i.e. a list of options).

   Below is an example of the corresponding description file for the above Advar template, followed by a screenshot of the Advar form in the Adhese interface:

```
{
"files": [{
"default": "",
"doc": "This will be used as logo",
"label": "A first image",
"key": "2nd"
},{
"default": "",
"doc": "This will be used as background",
"label": "A second image",
"key": "3rd"
}],
"advar": "advar_example.txt",
"fields": [{
"default": "",
"doc": "Select the source of your article",
"label": "Article source",
"type": "select",
"key": "<advar_source>",
"options": [{
"value": "afp",
"label": "AFP"
},{
```

```
"value": "belga",
"label": "Belga"
},{
"value": "reuters",
"label": "Reuters"
}]
},{
"default": "",
"doc": "Fill in the title of your article",
"label": "Title",
"type": "singleLineText",
"key": "<advar_title>"
},{
"default": "",
"doc": "Fill in the text of your article",
"label": "Text",
"type": "multilineText",
"key": "<advar_text>"
}
]}
```



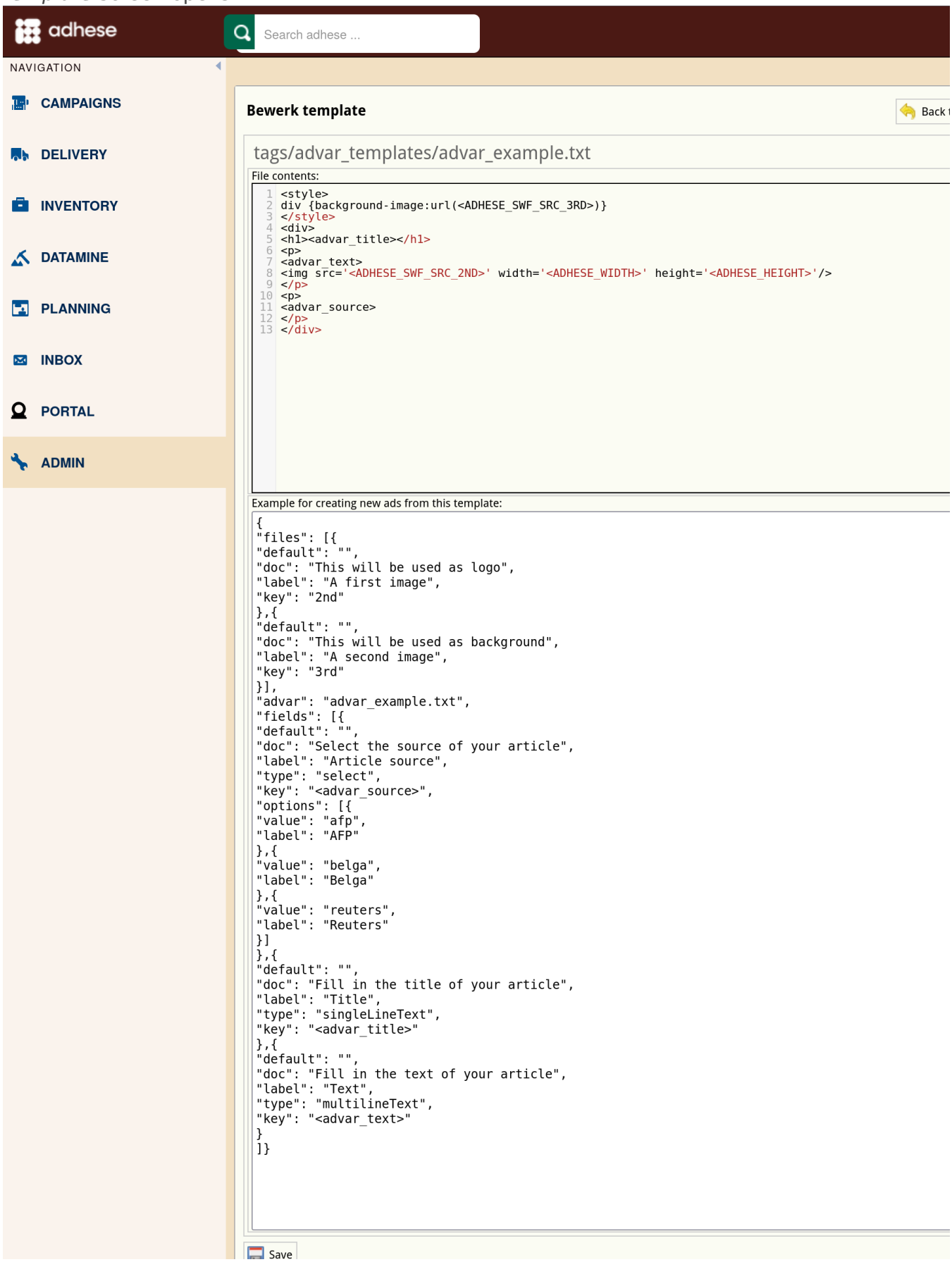
7. Click the *Save* button.

Edit an Advar template

To edit an Advar template:

1. Go to the *Administration* screen. Click *Admin* in the left navigation menu.
2. Click *Advar templates*.

3. In the list of Advar templates, click the Advar template you want to modify. The *Edit template* screen opens:



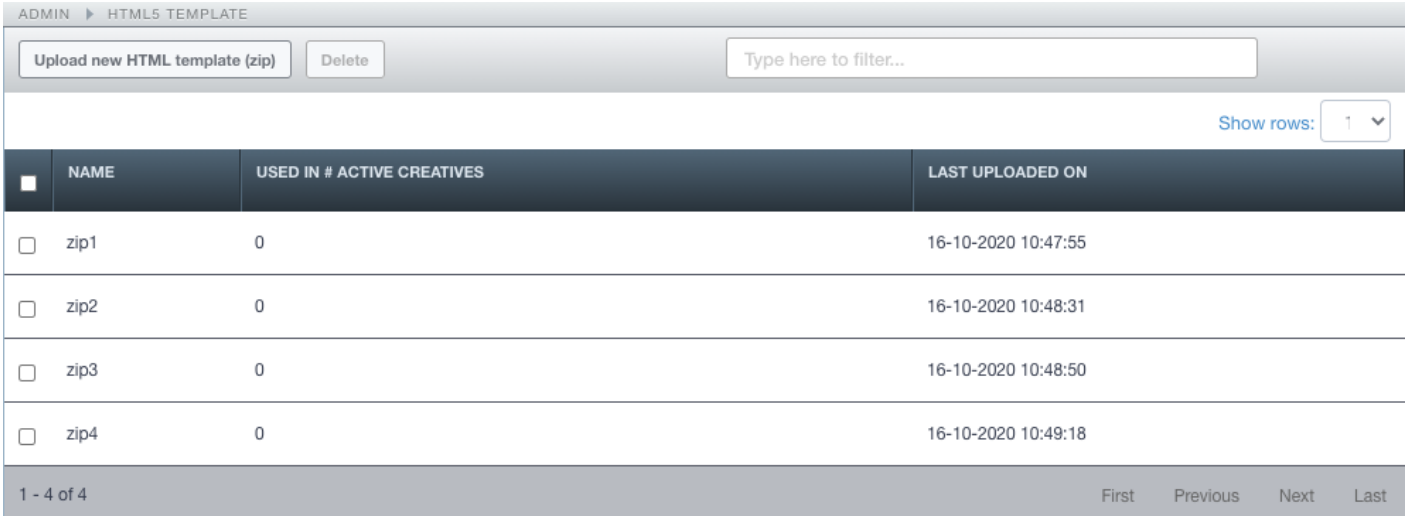
4. Change any of the Advar template's details.
5. Click *Save*.

HTML5 Templates

> It is not possible to use Adhese macros such as `[adheseReplace:xx]` or `<ADHESE_X>` from within an HTML5 template. Unlike in regular (advar) templates, these macros will not work.

An HTML5 template is a combination of a regular advar template and a folder with HTML, CSS and JS files. It allows you to use the form functionality of an advar template while storing code and optional media in a folder structure.

In *Admin > Formats and templates > HTML5 templates,* you can find all previously created templates in zip format. When you click on the link, the following screen will appear:



The following actions are available:

- Upload a new file: *click* the *Upload a new HTML5 template file (ZIP)* and choose a file to upload.
- Delete a template: *select* the file and *click* the *delete button*.
- Filter the list by entering a search term in the filter box.

Using HTML5 templates

Previously created HTML5 templates can be used by clicking the '*Add advar*' option in the creative tab and selecting the correct template in the '*Add Advar Template*' dropdown list.

Creating a new HTML5 template

> Ensure that the filenames only contain alphanumeric characters, dots, and underscores. Other characters may cause issues when Adhese processes the file!

Steps for creating and uploading a new template:

1. Locally, create a new folder in which to store all files
2. Choose a name for the template: **[name].zip**
3. Create a **descr.json** file and store it in the main directory of your folder. The content has to be constructed in the same way you would create a regular advar template. (See step 6 of creating a new advar)
4. Create an **index.html** file - which will be the starting point for the HTML5 ad - and store it in the main directory of your folder
5. Make sure to **incorporate clickthrough logic**. Adhese relies on it to track clicks correctly.
   You can do this by adding the following line of code to your **index.html** file. Adhese will add its click tracking URL to this variable, after which you can use it in the rest of your clickthrough logic.

   ```
   <script type="text/javascript">
     var clickTag = '';
   </script>
   ```

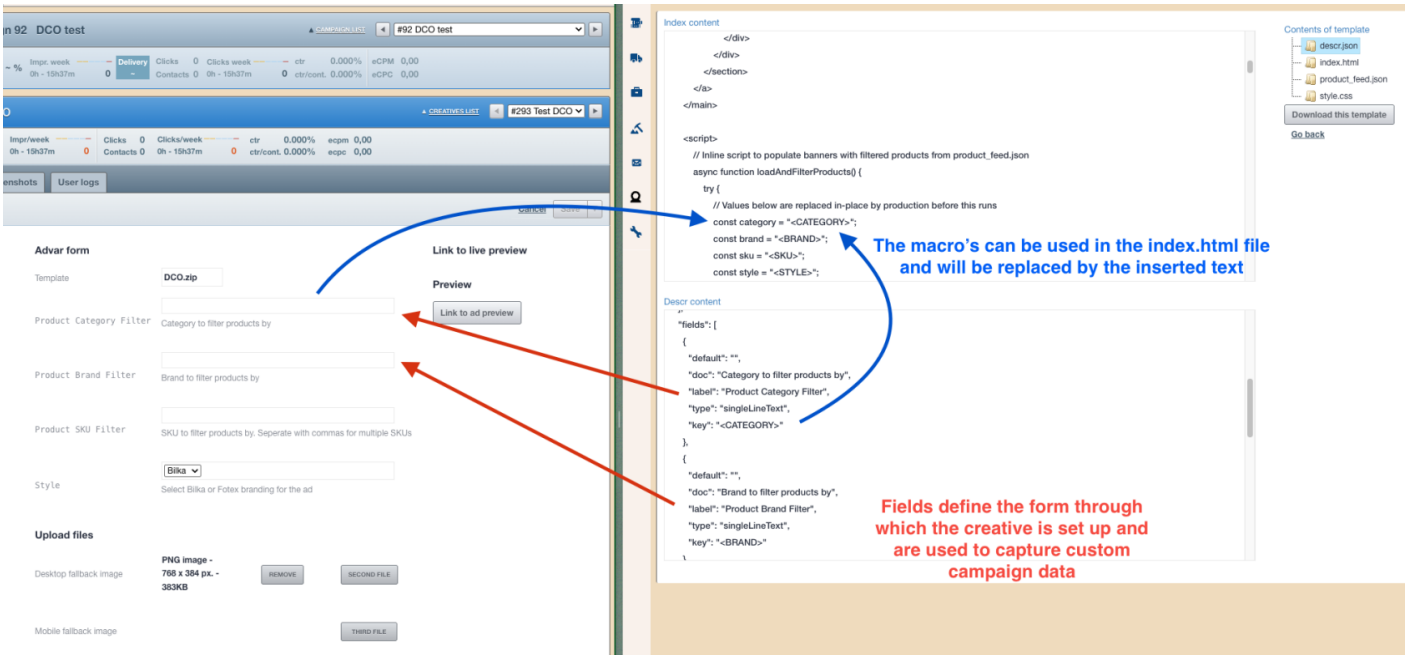
6. Optionally you can store CSS, JS and images in separate files & folders within the main directory.

   > All links in the HTML5 creative, such as the link to an image within the ad, need to use a relative path, for example: /graphics/ad-image.png or <img src="/graphics/ad-image.png">. This enables the ad to be self-contained and, therefore, to run independently or to render without a network connection. External libraries and web fonts can be an exception to this guideline.

7. Zip the content of the folder (not the folder itself) and **change the name of the zip** to the name of the template
8. Upload the template in the Adhese UI under *Admin > Formats and templates > HTML5 templates,*
9. Test the template by creating a creative and checking the preview of the banner

Retrieving text data from advar macro's

An advar template can contain fields that can be filled in while setting up a creative in the UI. Each of these fields has a macro that can be implemented in the index.html file. When the banner is served the macro's will be replaced by the text value provided through the creative.



Retrieving uploaded files

An advar creative can contain up to five extra files (count starts at 2) that will only be relevant for this specific creative. Templates that plan on using such files need to define the relevant fields in the description file:

```
"files": [
    {
      "default": "",
      "doc": "",
      "label": "Desktop fallback image",
      "key": "2nd"
    },
    {
      "default": "",
      "doc": "",
      "label": "Mobile fallback image",
      "key": "3rd"
    }
  ]
```

When served, each HTML5 creative will contain a **helpers** directory in which you can find **metadata.json**. This JSON can be used to determine if any uploaded files are available and what

their extensions are.

Example response:

```
{
  "ADHESE_2ND":"2nd.png",
  "ADHESE_3RD":"3rd.png"
}
```

The URL for metadata.json can be put together as follows:

```
 var metadata_url = location.protocol + '//' + location.host +
location.pathname.replace('/index.html', '') + '/helpers/metadata.json';
```

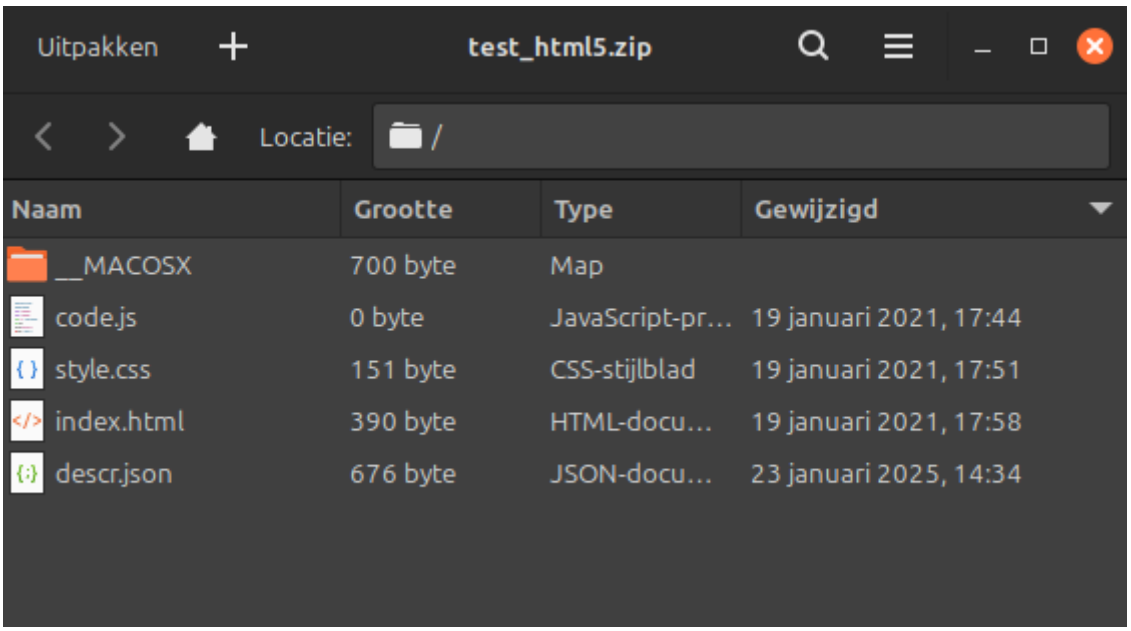
Each file will be stored in the same helpers directory. Once you know the extension of the file, you can put together the full URL:

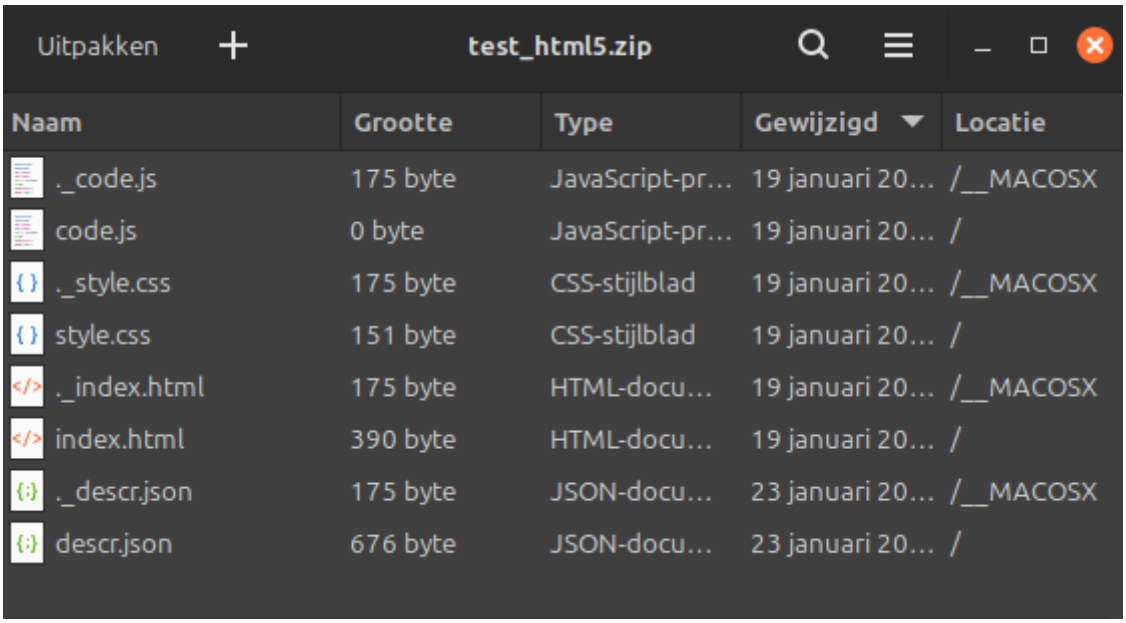
```
 var secondImage_url = location.protocol + '//' + location.host +
location.pathname.replace('/index.html', '') + '/helpers/' + '2nd.png';
```

Troubleshooting HTML5 Templates

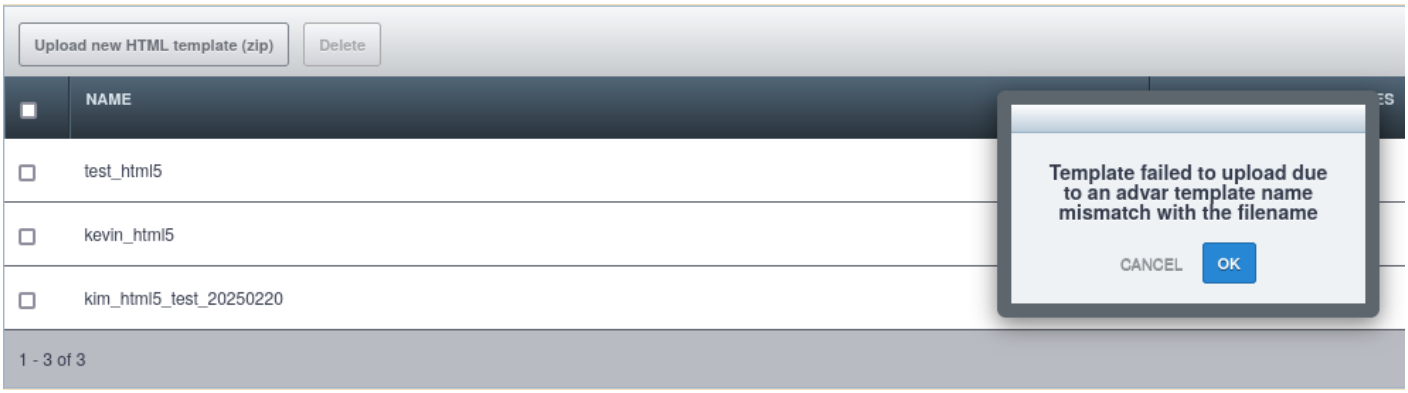
If an HTML5 template can't be uploaded for some reason, you'll get an error message:

In the case above, hidden files might block Adhese from properly unzipping the archive:





When creating an archive on Mac OS, a hidden __MACOSX folder might be created. These files must be removed for the upload to succeed (you may need to unzip, delete and re-zip the files).



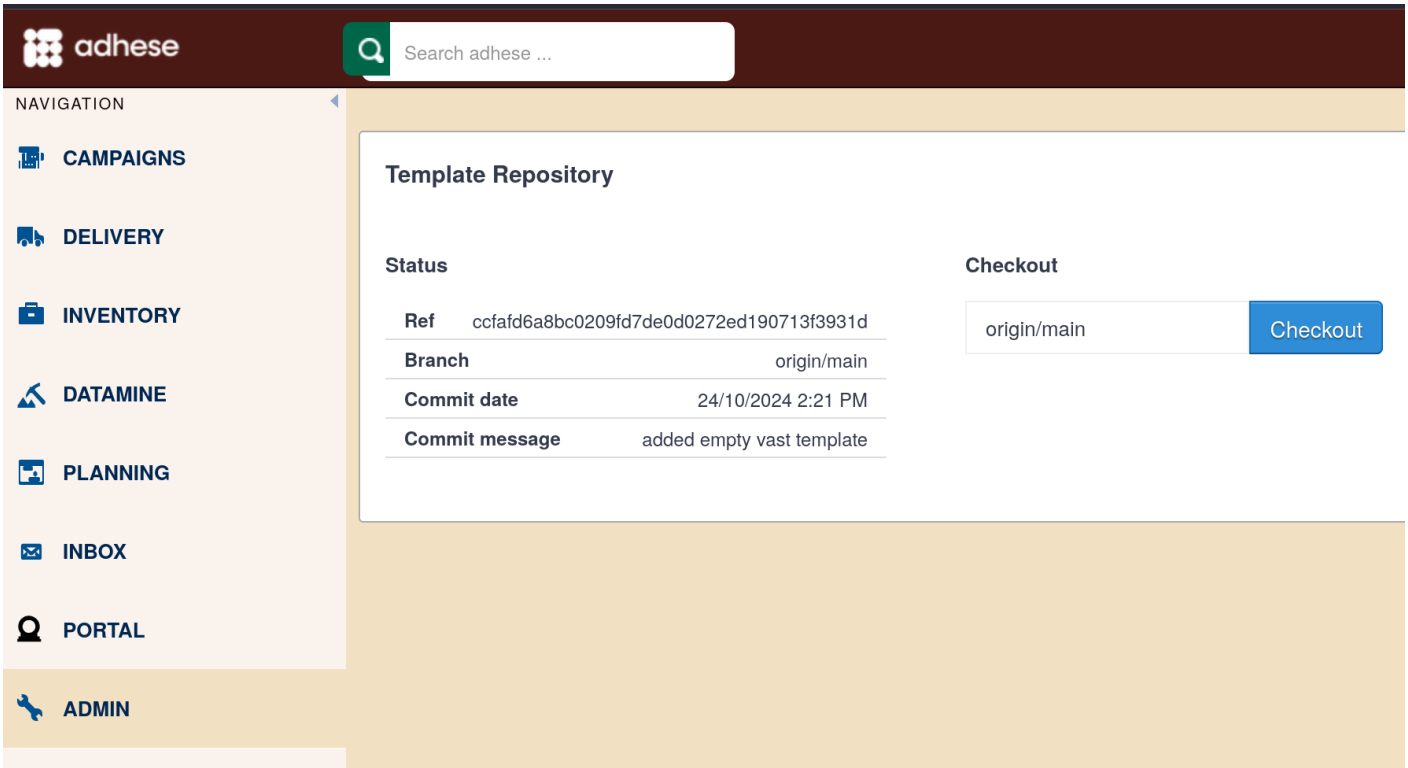
In the case above, the zip file has been renamed. If the zip file does not match the advar name in the descr.json file, rename the zip file to match the advar name in the descr.json file, and the archive can be uploaded.

Template Repository

The Template Repository lets you control Template files and Advar templates using a Git version control system. Once this option is enabled, you can

- store your templates in your own version control system
- maintain a detailed history of your changes
- effortlessly switch between different versions of your templates
- edit templates in your preferred IDE
- check out a specific branch in your Adhese account.

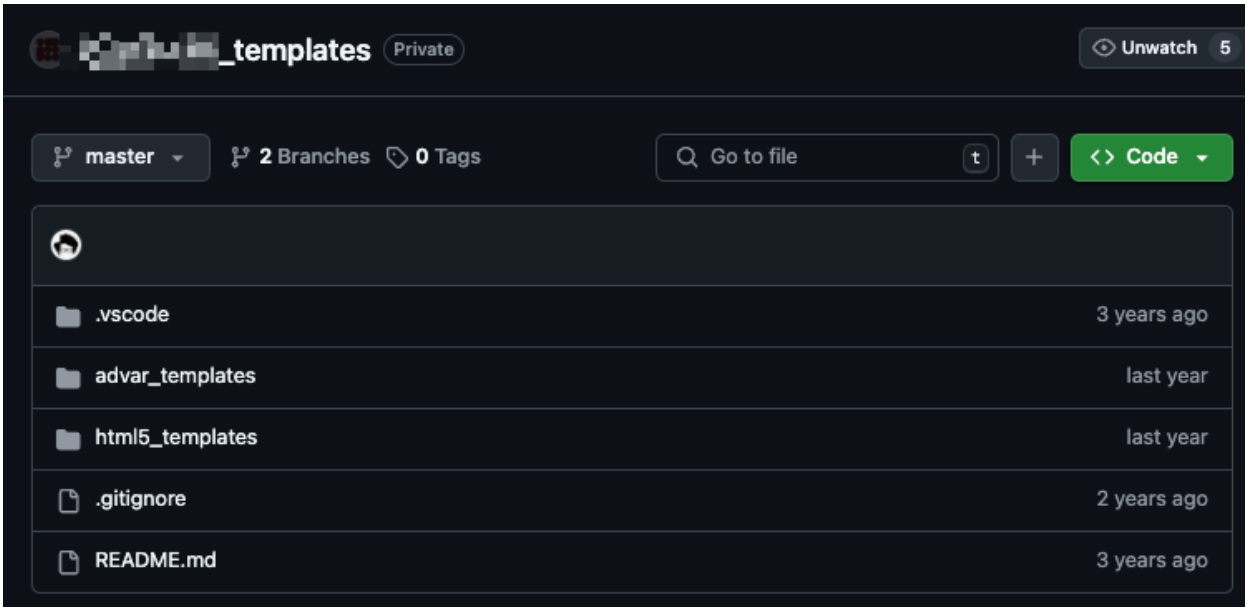
> Changes in the checked-out version on your Adhese account will be applied to **all relevant creatives** and will be pushed online with the next publish.



Directory Structure

The main directory of your repository can be used to store all 'regular' templates. In most implementations these are no longer used and can therefor be ignored. More relevant are the

advar templates and the 'HTML5' advar templates. Both types will be stored in separate folders: "**advar_templates**" and "**html5_templates**".



Advar templates

> More info about advar templates can be found here

Each advar template consists of 2 files:

1. The file that contains the actual response template. This file can be made out of HTML, JSON or XML code. It will depend on the type of integration the template will be used for. The extension can be .txt or something that matches the content of the template.

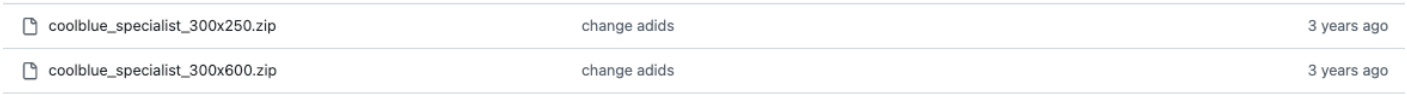
2. A description file that contains the form used to create an advar creative. This file will contain specific JSON markup. More info on this can be found here. The file needs to have **the same name as the first file, followed by the extension .descr** .

vast3_ad.txt	initial commit for with existing templates	4 years ago
vast3_ad.txt.descr	initial commit for with existing templates	4 years ago

HTML5 templates

> More info about HTML5 templates can be found here

HTML5 templates are stored as compressed folders using the .zip extension. Each HTML5 template has their own HTML, CSS and JS structure within that folder and need to have a description file named descr.json in the main directory.



Usage

The Template Repository screen shows two panels. On the left, you can see a summary of the Git commits currently used. It contains the Git hash, the branch, and the date and message when it was committed. On the right, you see a text field and button to change the commit for checkout:

1. Enter branch name or Git hash to use (e.g. origin/master)
2. Press *Checkout* button

The specified Git commit will be checked out. All the template changes will be applied in the next publish phase.

Activation

To activate this option, please get in touch with our Support department. You will also need to provide the following information:

- the URL of your Git repository, this needs to be accessible from the outside (e.g. git@github.com:adhese/my_template_repo.git)

We will send you the public SSH key that you need to add to your Git configuration to allow us access to the repository you would like to use for managing your templates. If you use GitHub, add this as an SSH key to Your Repo > Settings > Deploy Keys.

Adhese Parameters

Server-side request parameters

The ad server replaces the following list of parameters with the request information for each personalised request. The parameters can be included in any template, including Advar templates or third-party code.

Parameter	**Description**
[adheseExpand: <ID>]	
[adheseReplace: <ID>]	Is replaced by the values sent with the corresponding ID from the request.
[adheseLogID]	A unique number used for reporting.
[adheseRequestData]	All parameters used in the ad request.
[adheseRequestDataFlat]	Contains the full request as it is sent to the ad server but replaces all semicolon-separated values by a /prefixValue/ clause.
[adheseSetExpandPrefix: <prefix>]	
[adheseTimestampNowMs]	Unix timestamp at the moment of sending the response, the number of milliseconds since 1970-01-01 0:00:00.
[adheseAdditionalRequestParameters:<target>]	This parameter can contain additional parameters that are added to the targets of the request, for example, dmADV <ADHESE_ADVERTISER_ID>;OR <ADHESE_ORDER_ID>.
[adheseEnv:<KEY>]	This parameter is replaced by the environment variable <KEY> of the request, for example [adheseEnv:HTTP_HOST]. See http://en.wikipedia.org/wiki/Common_Gateway_Interface for an overview of all environment variables.
[adheseReplace:SL]	The value of the string identification for the requested position.
[adheseReplace:A2]	The value of the identification cookie.
[adheseDomain:ad_host], [adheseDomain:click_host], [adheseDomain:pool_host], [adheseDomain:track_host]	Contains the incoming host for the request. Usefull for 1st domain implementations with multiple domains.

Example creative template:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Demo Image Template</title>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script>
        const source = "[adheseReplace:SL]";
        const domain = source.split("_")[1];
        document.querySelector("html").classList.add(domain);
    </script>
    <style>
          body {
            margin: 0;
            overflow: hidden;
         }
        .br #product-image {
            border-radius: 20px;
        }
    </style>
  </head>
  <body>
    <a href=" %c<CLICKURL>" target="_blank">
        <img id="product-image" src="<ADHESE_SWF_SRC_2ND>" width=<ADHESE_WIDTH_LARGE>
height=<ADHESE_HEIGHT_LARGE> alt=<ADHESE_POSITION>/>
    </a>
  </body>
  </html>
```

Macros for templates and Advar templates

When working with **advar templates** & advar creatives, you need to **use the 2ND (and higher)** macros when adding macros related to **uploaded creative files**. Examples are <ADHESE_EXT_2ND>, <ADHESE_SWF_SRC_2ND>

If you encounter invalid JSON responses (e.g., quotes in the campaign name), use the URI-encoded versions of the macros.

Macro	URIENCODED	Description
<ADHESE_ADSPACE_COMMENT>	<ADHESE_ADSPACE_COMMENT_URIENCODED>	The comment of the booking.
<ADHESE_ADSPACE_END>		The end date of a booking.
<ADHESE_ADSPACE_ID>		The unique ID of the booking.
<ADHESE_ADSPACE_KEY>	<ADHESE_ADSPACE_KEY_URIENCODED>	The external reference or key of a booking.
<ADHESE_ADSPACE_START>		The start date of a booking.
<ADHESE_ADVERTISER_ID>		The ID of the selected Advertiser company
<ADHESE_ALT_TEXT>	<ADHESE_ALT_TEXT_URIENCODED>	The content of the alt text field.
<ADHESE_BODY>	<ADHESE_BODY_URIENCODED>	The body of the creative (if available).
<ADHESE_CLICK_TAG>	<ADHESE_CLICK_TAG_URIENCODED>	The click tag of a creative (including URL).
<ADHESE_CONFIGURABLE_TRACKER_URL>	<ADHESE_CONFIGURABLE_TRACKER_URL_URIENCODED>	If configured, the tracking URL that has been configured in the configuration.
<ADHESE_CREATIVE_ID>		The unique ID of the link between the booking and the creative.

<ADHESE_CREATIVE_NAME>	<ADHESE_CREATIVE_NAME_URIENCODED>	The name of the creative.
<ADHESE_DELIVERY_MULTIPLES>	<ADHESE_DELIVERY_MULTIPLES_URIENCODED>	The delivery multiples value of the booking.
<ADHESE_DELIVERY_MULTIPLES_GROUP_ID>	<ADHESE_DELIVERY_MULTIPLES_GROUP_ID_URIENCODED>	The group ID for the delivery multiples, if needed.
<ADHESE_DURATION>, <ADHESE_DURATION_2ND>, <ADHESE_DURATION_3RD>, <ADHESE_DURATION_4TH>, <ADHESE_DURATION_5TH>, <ADHESE_DURATION_6TH>		The duration of the uploaded video files in seconds.
<ADHESE_DURATION_MS>, <ADHESE_DURATION_MS_2ND>, <ADHESE_DURATION_MS_3RD>, <ADHESE_DURATION_MS_4TH>, <ADHESE_DURATION_MS_5TH>, <ADHESE_DURATION_MS_6TH>		The duration of the uploaded video files in milliseconds.
<ADHESE_EXT>, <ADHESE_EXT_2ND>, <ADHESE_EXT_3RD>, <ADHESE_EXT_4TH>, <ADHESE_EXT_5TH>, <ADHESE_EXT_6TH>	<ADHESE_EXT_URIENCODED>, <ADHESE_EXT_2ND_URIENCODED>, <ADHESE_EXT_3RD_URIENCODED>, <ADHESE_EXT_4TH_URIENCODED>, <ADHESE_EXT_5TH_URIENCODED>, <ADHESE_EXT_6TH_URIENCODED>	The file extension of each uploaded file.
<ADHESE_EXTRA_FIELD_1>	<ADHESE_EXTRA_FIELD_1_URIENCODED>	The content of the extra field 1.
<ADHESE_EXTRA_FIELD_2>	<ADHESE_EXTRA_FIELD_2_URIENCODED>	The content of the extra field 2.
<ADHESE_FORMAT>	<ADHESE_FORMAT_URIENCODED>	The format of the booking.
<ADHESE_HEIGHT_LARGE_3RD>, <ADHESE_WIDTH_LARGE_3RD>		The dimensions of the third file that is uploaded.
<ADHESE_HEIGHT_LARGE_4TH>, <ADHESE_WIDTH_LARGE_4TH>		The dimensions of the fourth file that is uploaded.
<ADHESE_HEIGHT_LARGE_5TH>, <ADHESE_WIDTH_LARGE_5TH>		The dimensions of the fifth file that is uploaded.
<ADHESE_HEIGHT_LARGE_6TH>, <ADHESE_WIDTH_LARGE_6TH>		The dimensions of the sixth file that is uploaded.
<ADHESE_LIB_ID>		The unique ID of a creative in Adhese. This ID can be used to make JavaScript functions unique.
<ADHESE_ORDER_ID>		

<ADHESE_ORDER_NAME>	<ADHESE_ORDER_NAME_URIENCODED>	The name of the campaign.
<ADHESE_ORDER_PRIORITY>	<ADHESE_ORDER_PRIORITY_URIENCODED>	
<ADHESE_POOL_PATH>	<ADHESE_POOL_PATH_URIENCODED>	The location of where the ad is located on the server.
<ADHESE_POSITION>	<ADHESE_POSITION_URIENCODED>	The position of the booking.
<ADHESE_PRIORITY>	<ADHESE_PRIORITY_URIENCODED>	The priority of the campaign.
<ADHESE_SHARE>	<ADHESE_SHARE_URIENCODED>	Deprecated.
<ADHESE_SWF_SRC>, <ADHESE_SWF_SRC_2ND>, <ADHESE_SWF_SRC_3RD>, <ADHESE_SWF_SRC_4TH>, <ADHESE_SWF_SRC_5TH>, <ADHESE_SWF_SRC_6TH>	<ADHESE_SWF_SRC_URIENCODED>, <ADHESE_SWF_SRC_2ND_URIENCODED>, <ADHESE_SWF_SRC_3RD_URIENCODED>, <ADHESE_SWF_SRC_4TH_URIENCODED>, <ADHESE_SWF_SRC_5TH_URIENCODED>, <ADHESE_SWF_SRC_6TH_URIENCODED>	The target URLs of the uploaded files, will be empty in case not uploaded.
<ADHESE_TAG>	<ADHESE_TAG_URIENCODED>	The complete HTML code of an ad: object/embed code in case of a .swf file, JavaScript in case of third-party code, or img link tags in case of a static image.
<ADHESE_TEMPLATE_CODE>	<ADHESE_TEMPLATE_CODE_URIENCODED>	The code of the template for the position of the booking.
<ADHESE_TEMPLATE_CODE_EXPORT>	<ADHESE_TEMPLATE_CODE_EXPORT_URIENCODED>	The export code of the template of the creative.
<ADHESE_TRACKING_URL>	<ADHESE_TRACKING_URL_URIENCODED>	The URL for tracking 3rd party impressions.
<ADHESE_URL>	<ADHESE_URL_URIENCODED>	The target URL or landing page of the ad, as determined by the user.
<ADHESE_IMPRESSION_TRACKING_URL>	<ADHESE_IMPRESSION_TRACKING_URL_URIENCODED>	The URL to count trackable impressions
<ADHESE_VIEWABLE_TRACKING_URL>	<ADHESE_VIEWABLE_TRACKING_URL_URIENCODED>	The URL to count viewable impressions.
<ADHESE_WIDTH>, <ADHESE_HEIGHT>		The dimensions of the first uploaded file.
<ADHESE_WIDTH_LARGE>,<ADHESE_HEIGHT_LARGE>		The dimensions of the second file that is uploaded.

Macros in Advar fields

If you want to fill in a macro in a field of an Advar template during creative creation, i.e. `pool-demo.adhese.com?bg_sourceid=<ADHESE_LIB_ID>` to be filled in a Landing Page field, you need to allow the macro to be replaced within the Advar template itself.

For example:

```
{
            "default": "<ADHESE_LIB_ID>",
            "doc": "replacementtest",
            "label": "NA",
            "type": "singleLineText",
            "key": "<recu-open_ADHESE_LIB_ID>"
}
```

Where <recu-open_[MACRO]> allows the macro to be replaced in the Advar field.

> For the macro replacement to work, the account configuration needs to be changed. Please contact Support when you want to make use of this feature.

Stack sequence

Sequencing

The sequence parameter can be used in advar templates to sort ads within the request response, and is therefore only relevant for **stack implementations** where multiple ads are returned for one placement. Some examples are:

- VAST Adpods
- Digital Out Of Home playlists
- Native advertising

Sequence suggestions are optional, and when implemented, Adhese will do its best to place the ad in the desired position within the response.
You can enter any number as a sequence suggestion, but obviously, it only makes sense to use unique numbers. An option is to use Adhese macros such as `<ADHESE_LIB_ID>` to implement sorting based on the creative ID.

> The sequence value is **stripped from** the response when the adpod or stack response is rendered.

Templates

VAST

If `sequence=0`, the ad is preferred to be placed first in a pod. If `sequence=-1`, the ad prefers to be placed last in a pod. If `sequence=1` it prefers the second place and so on. This ordering happens after the ads are selected, so there is no guarantee that an ad with `sequence=0` will always be first in the pod (unless if it is the highest priority of all possible options).

```
<Ad sequence="0">
VAST ad implementation
</Ad>
```

JSON

> The sequence field must be added to the **main structure** of the JSON template. The order logic will not work if the field is added inside a nested object in the JSON structure.

```
{
  "sequence": 0,
  "example field": "some value"
}
```

Runtime time string replacement and encoding

Adhese contains a number of scripts that can be added to any kind of template and will be interpreted at runtime. This allows you to create complex templates where variables can be transformed or encoded via the adheseScript tags.

The scripts are useful for URL encoding, JavaScript escaping, string replacement and other similar tasks.

Tags in the format `[adheseScript...]` have a corresponding `[/adheseScript]`. Each tag applies some modifications to the text in between these tags.

Note that it is possible to nest these tags to apply multiple operations to the same text. The tags are then applied one at a time, with the inner ones being applied first and then the outer ones.

Example of nested tags:

```
[adheseScriptBase64Decode]
  [adheseScriptReplace:-:l]he--o wor-d[/adheseScript]
[/adheseScript]
```

Let's look at some tags in more detail.

Non-escaping script tags

adheseScriptReplace

Searches the enclosed input text for the first parameter and replaces it with the second parameter. Search and replace is done with literal text, not with regex.

```
[adheseScriptReplace:-:l]he--o wor-d[/adheseScript]
```

This results in

```
hello world
```

adheseScriptLower

Puts the enclosed input text in lowercase.

```
[adheseScriptLower]FOOBAR[/adheseScript]
```

results in

```
foobar
```

adheseScriptBase64Decode

Applies Base64 decoding to the enclosed input text.

```
[adheseScriptBase64Decode]What a nice string of characters[/adheseScript]
```

results in

```
V2hhdCBhIG5pY2Ugc3RyaW5nIG9mIGNoYXJhY3RlcnM=
```

adheseScriptUriEncode

Applies UriEncoding to the enclosed input text.

```
[adheseScriptUriEncode]What a nice string of characters[/adheseScript]
```

results in

```
What%20a%20nice%20string%20of%20characters
```

Escaping script tags

By default, some escaping is applied to any `[adhese...]` tags. If multiple tags are nested inside each other, the default escaping will only be applied once, after the final tag is executed.

The default escaping does the following:

```
' ? \'
" ? \"
\ ? \\
]]> ? ]]]]><![CDATA[>
\b \t \n \r \f
unicode characters outside the range 32, 0x7f
```

This should be sufficient for the majority of use cases. However, there may be edge cases where additional control over escaping is required. This can be achieved by utilising one of the tags outlined below.

For example:

- When used within XML without CDATA tags (although we recommend the use of CDATA tags instead of more complex escaping techniques)
- When multiple payloads are nested within each other, a single round of escaping is not enough. The default escaping process would protect against the injection of malicious code into the JavaScript, but an attacker would be able to inject malicious code into the inner HTML. To ensure security, this tag must be escaped twice.

  ```
  var foo = "<img src=\"[adhese...]\" >";
  ```

- When `[adhese...]` tags are placed directly inside HTML without being enclosed in `'` or `"` By default `<` and `>` are not escaped, so an attacker could insert any HTML they want.

  ```
  <h1>[adhese...]</h1>
  ```

If the outermost tags apply to escape (as in one of the tags listed below, except for adheseScriptEscape), then the default escaping is disabled.

If JSON is requested, additional JSON escaping will be applied. This additional escaping is the same as the default escaping described above, with the exception of single quotes and `]]>`, which are not escaped. Additional escaping is applied even when the default escaping is turned off.

adheseScriptEscapeXMLCDATA

```
]]> ? ]]]]><![CDATA[>
```

Useful in XML CDATA tags when the default escaping breaks things by escaping too much.

adheseScriptEscapeXML

Useful for escaping inside XML that does not use CDATA tags.

See https://commons.apache.org/proper/commons-text/javadocs/api-release/org/apache/commons/text/StringEscapeUtils.html#escapeXml11(java.lang.String) for details.

adheseScriptEscapeJSStringWFS

Equivalent to the default escaper but doesn't escape `]]>`

Useful for js. Doesn't escape `/` so don't use it directly in a js regex replace

adheseScriptEscapeJSString

Equivalent to adheseScriptEscapeJSStringWFS but it also escapes `/`

see also: https://commons.apache.org/proper/commons-text/javadocs/api-release/org/apache/commons/text/StringEscapeUtils.html#escapeEcmaScript(java.lang.String)

adheseScriptEscapeHTML

applies https://commons.apache.org/proper/commons-text/javadocs/api-release/org/apache/commons/text/StringEscapeUtils.html#escapeHtml4(java.lang.String)

but also escapes `' ? &#39;`

adheseScriptEscapeNone

Escapes nothing, which is useful for JSON fields that are not embedded into something else (as the separate JSON escaping is enough there).