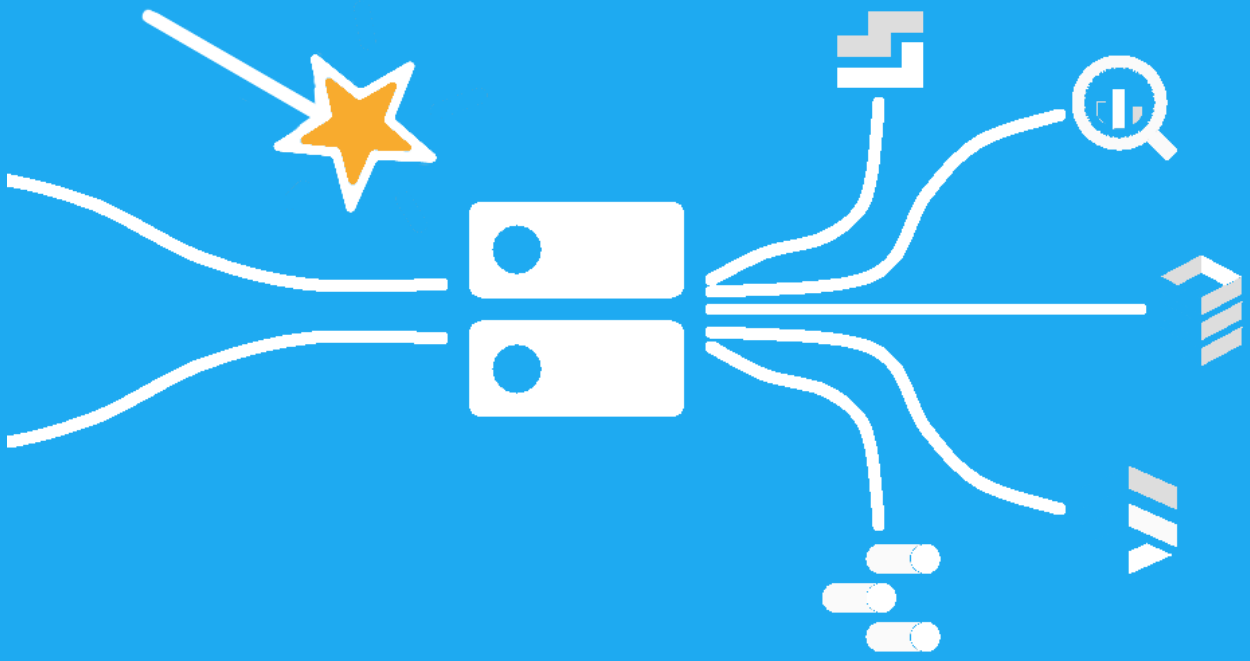
The Unofficial Manual

Sample

Google Tag Manager Server Templates

Use, Understand & Develop Your Own Templates



Markus Baersch

2nd edition | April 2022

Special Bonus Chapter: Tag Template for client-side Google Tag Manager

Foreword

ITP, ETP and other browser tracking restrictions make client-side tracking more and more difficult. The use of server-side Google Tag Manager is often seen as some kind of universal solution. However, the server-side Google Tag Manager solves many but not *all* problems. More importantly, server-side GTM, which has only recently left beta, has a lot of potential, which can currently be fully unleashed only through individual development of templates for clients, tags and variables. Because the current offer of predefined tags, clients and variables is very limited. Although Template Gallery is growing steadily, the ability to develop your own templates is an important step if you do not want to be dependent on third parties.

With this book, Markus Baersch has created a simple introduction to template development for server-side Google Tag Manager, which focuses on necessary theoretical information and a strong practical part. Many topics are addressed for an easy and quick start, which enables the reader to deal with the relevant topics in more detail.

Marcus Stade

2nd Edition Foreword

It did not even take half a year for such significant innovations to occur in ssGTM's feature set that an update of this e-book became necessary. The closed gap concerns one of the key differences between this solution for establishing a dedicated tracking endpoint and many others: The enrichment of data at the server before it is passed from tags to vendors.

Two key things have been added to this:

1. Promises, which enable the use of APIs, and
2. Firestore as a separate persistence (beyond Big Query) for fast reading and writing access

These additions have led to minor updates and changes in this e-book, in addition to a completely new chapter on data enrichment. For example, in the tips, the section *Respond via returnResponse()* has been added.

Moreover, if you compare the sample codes from the first edition with the second, you will notice that the previous callback functions of API calls like *sendHttpGet()* in existing codes have *not* been replaced by Promises (for more on Promises, see [38]). This is because the adaptation in the Google APIs does not mean that the previous methods no longer work, nor does it bring any real advantage in the examples from this e-book. New examples in the added chapter on data enrichment demonstrate the advantages in one spot, so that one does not have to work through the whole e-book again to get to the added knowledge of the second edition. Have fun with ssGTM and its new possibilities!

Markus Baersch, April 2022

Introduction

Serverside GTM without templates is like a race car without a track

Since the server-side variant of Google Tag Manager has been available, its use has been primarily focused on the Google universe: Pre-built clients focus entirely on Google Analytics formats and secure delivery of the client-side container and tracking scripts. Tags send data to Google Analytics or Ads. For everything else, there's a rudimentary template for sending requests. That's pretty much it.

To bring server-side Google Tag Manager (from here on short: "ssGTM") to its full potential, much more is needed depending on the intended use. Data collection in other formats, forwarding to services other than Google Analytics or Ads: All this is mostly only possible with custom templates. You have to create them yourself or use templates for clients, tags or variables provided by third parties.

The Community Gallery [1] for Google Tag Manager already provides some templates that can be imported and used. These solve tasks such as connecting systems (e.g. ActiveCampaign, Klavyio, Mailchimp or HubSpot), server-side tracking for Facebook and other services.

Existing templates for variables allow processing or adaptation of received data for targeted further processing in tags. Clients, on the other hand, are currently completely missing and can only be imported manually if a template exists elsewhere.

Although the rather thin supply situation is becoming more multifaceted over time, individual requirements for receiving, processing and forwarding incoming data usually require specially tailored templates.

This book uses practical examples to show how to create and use your own templates for tags, clients and variables.

About the author

Markus Baersch is the managing director of gandke marketing & software gmbh (www.gandke.de) in Mönchengladbach (Germany. They support customers from every market segment in making marketing and ongoing optimization of their shops or B2B websites successful. For about 15 years, mainly digital analytics, tag management and SEM have been part of his professional routine. Together with Michael Janssen, he has been running the web analytics podcast "beyond pageviews" on termfrequenz.de [2] since 2016.

Designing and implementing client-side and server-side tracking and tagging with and without Google Tag Manager has been a focus area for years - both operationally and in the form of training, workshops and talks. Since the public availability of ssGTM as a beta in August 2020, it has quickly become the preferred platform for more complex tracking setups. This has resulted in a number of templates for the client-side and server-side tag manager, some of which are also available in the Community Gallery.

Status, changes and additions

The first edition of this book was written in fall of 2021; coincidentally parallel with the end of the ssGTM beta phase. ssGTM beta had not been publicly available for a year and a half at that point. Changes are therefore very likely depending on when you read this guide.

The second edition is based on the functional state of April 2022. It includes significant changes, described in the foreword.

Updates on the capabilities presented in this book and other resources related to Tag Manager can be found on the author's website [3] and on the YouTube channel [4].

Custom Templates in Google Tag Manager

Custom templates in Google Tag Manager ("GTM") are nothing fundamentally new. The concept is already known from the “normal” client-side Google Tag Manager and in ssGTM it is very similar.

A template is used to add new elements to GTM, which are available for selection when a tag, variable - or, in the case of ssGTM, a client - is created.

How it works

To create your own template, GTM offers its own template **editor**. Templates can be created, provided with an **interface** for defining parameters ("fields"), tested and provided with the necessary **permissions**.

For **coding** templates, JavaScript is used in a so-called "sandbox". This means that the code of every template is operated in a secure mode at runtime, in which only functions and resources that are made available by the sandbox can be used. These restrictions are intended to ensure both security and robustness of the code, as well as to create transparency about what information, for example, a tag may process and where it is allowed to send data.

In these aspects, templates for operation in the browser do not differ from those developed for ssGTM. In detail, however, there are clear differences.

This is why own templates on the server are indispensable

An essential point regarding templates is that they're different in ssGTM than they are in the browser: There are no alternatives. While in client-side GTM you can use the full potential of JavaScript by coding *user-defined JavaScript variables* and script code in *user-defined HTML tags*, the situation on the server is much more restrictive. All desired processing steps when receiving incoming requests (in clients) and passing them on to tags are restricted by the existing properties of these elements. Only data from the ssGTM event model or incoming request data is passed on. The data either remains unchanged or is transformed, for example, by using variables.

This difference between server and web GTM is quite serious: Unlike web GTM, the server-side GTM cannot simply use JavaScript to enable transformations. There is only a limited set of APIs that can be used in templates to

- receive information (e.g. request cookies from the browser),

- transform and
- (send request, set cookie, etc.).

Many standard sandbox functions available in templates for Web containers cannot be used on the server.

The diagram from the introduction to how SSTM works [5] shows the essential role of clients and tags in the ssGTM in the data processing chain.

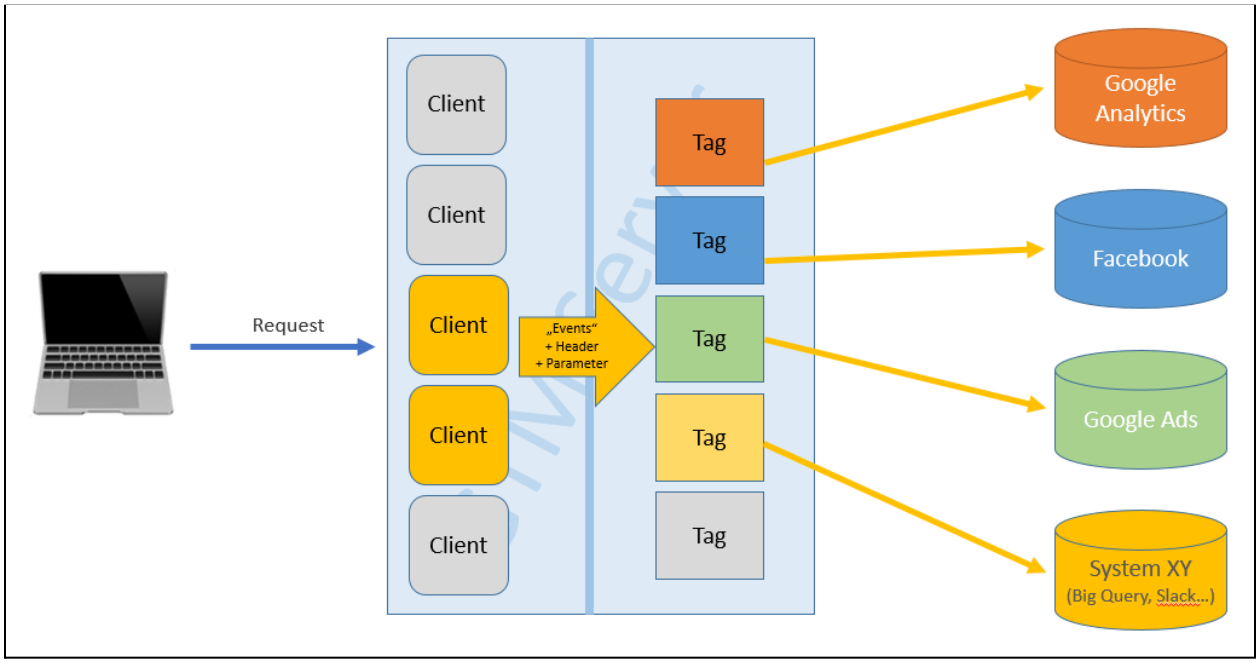


Figure 1: Functionality of ssGTM

In one sentence: Requests are received, transferred by *clients* (which can process this incoming format) into a standard event model and then handed over to 1:n *tags,* which then send the data in the appropriate formats to tracking services.

As soon as a request format other than "Analytics" (Universal Analytics, GA4 or the corresponding versions of the Measurement Protocol) is to be processed, a suitable client is needed.

The same applies to tags that are usually responsible for passing on data to tracking services such as Google Analytics, Facebook, Matomo or other recipients like Big Query databases or others.

In practice, this means: Even for “small” things you might need your own template. Transformations or additions of data for a specific recipient (in form of a tag) can fortunately be

done with variables - even using existing templates. For this, however, the appropriate tag must be available.

Templates for clients, on the other hand, are always necessary for alternative forms of data collection (usually in the browser) and prcessing incoming hits to ssGTM in a proprietary format. This can be a self-defined format or requests from an existing system like Piwik PRO, Matomo, Plausible or other analytics solutions, to remain in the web analytics context.

APIs from CRM systems, marketing automation, customer data platforms, apps, access control systems, or any other potential data sources like IoT devices can also be used to send data to an ssGTM endpoint for further processing.

Fortunately, there are already some templates available for various purposes. Therefore, as a first step, it is always a good idea to have a look around the *Community Template Gallery.* This can be done either in a "complete catalogue" [1] on the web - or directly in GTM when creating new elements.

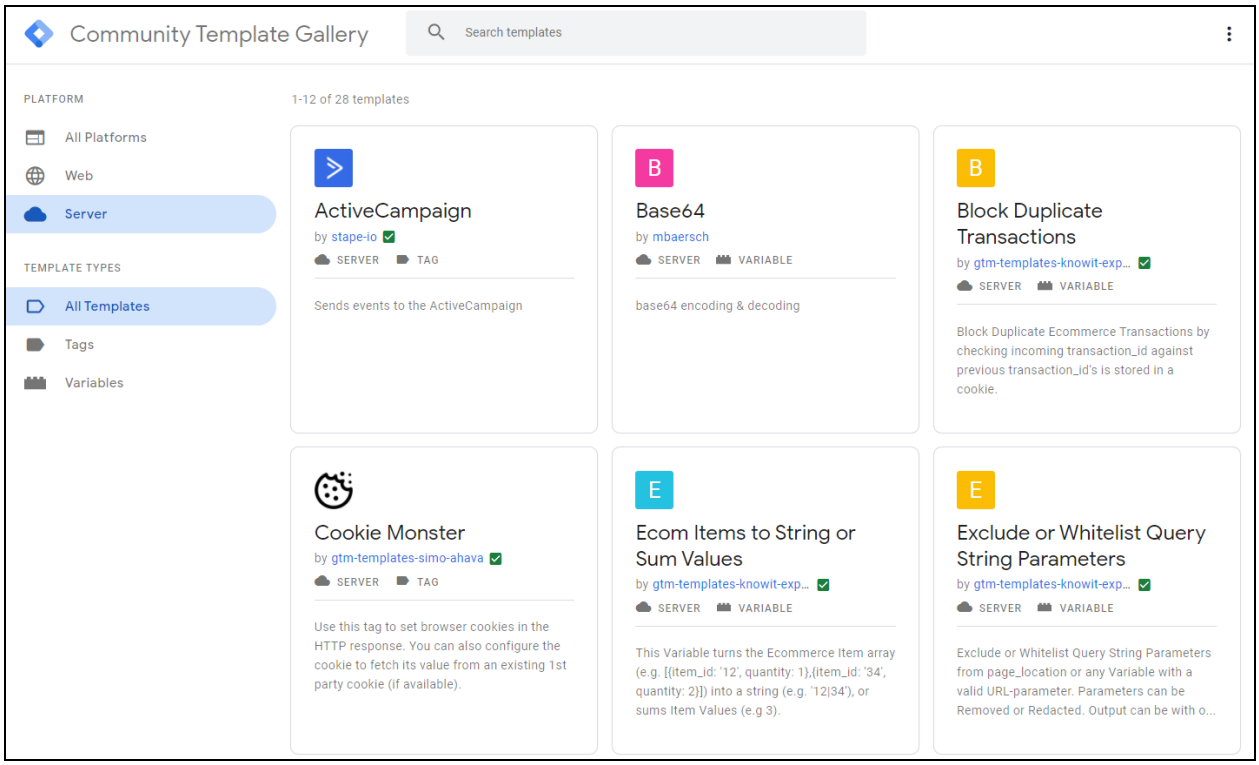


Figure 2: Community Template Gallery

Using custom templates

If a new element such as a tag or a variable (clients are currently not available in the gallery) is created in GTM, the type can be selected from a list. Above existing items there is a link for accessing available templates from the community.

Import from the Gallery

Existing templates from the Gallery are offered for import. For each type, in addition to information about the author, there is a link to the repository and the author's website and precise information about what permissions are required.

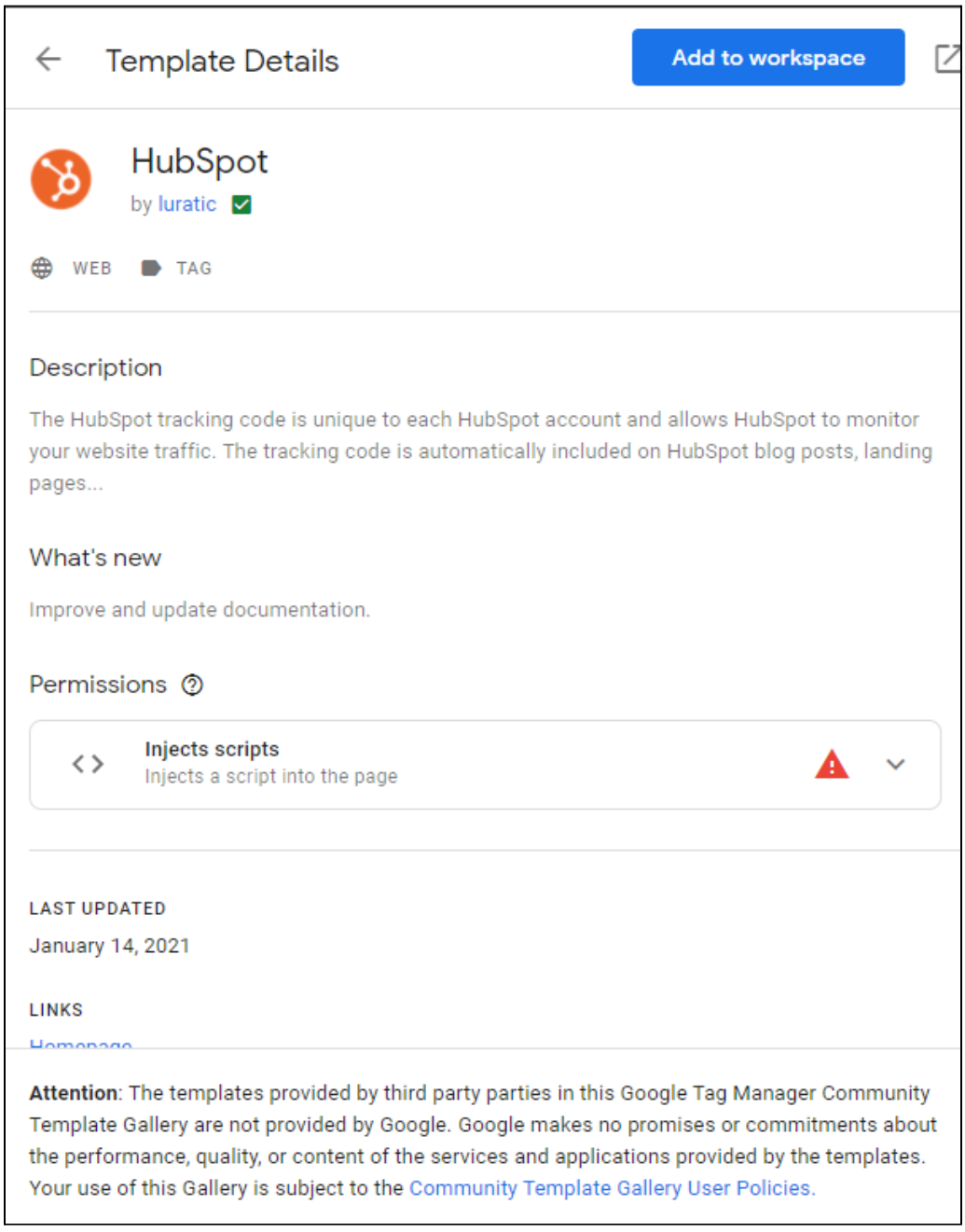


Figure 3: Details of a template from the Community Template Gallery

With one click, the template can be imported and then used in your GTM container.

Another advantage of templates is that they are usually maintained and updated if, for example, the format of the connected service changes. When updates are available, the GTM actively informs and offers the import of new templates after checking the changes.

As the example above shows, a tag for connecting a service does not necessarily have to be provided by the respective operator itself. Instead, most templates currently originate from practice: Someone wanted or needed to connect service X (in this case HubSpot), developed a template for this purpose, and made it available to other users via the Template Gallery.

Manual import

Even for clients there are already some candidates. However, these are (still) on GitHub and other platforms, or may be offered by the author on their own website. If you search GitHub for "*tag manager client template*", you will find 23 repositories at the time of writing (April 2022).

Also as "Gist" (the little brother of repositories) some further templates for clients can be found there; among others the client *200 Qapla'* [6] (more about that in a moment). All templates consist, at their core, of a single file containing all the necessary details about the template, the code and field definitions, tests created and so on. This is always called *template.tpl* for standard templates from the Gallery.However, as in the example of *200 Qapla',* it can also be given another name as long as the extension *.tpl is* retained.

If you download the file (e.g. via the "*Raw*" button above the code and "*Save as...*" in the browser) onto your own computer, you can then import the template manually in ssGTM.

For this purpose, a new template is created under "Templates" in ssGTM in the corresponding area (here: Client templates) and the import of the template is started via the "..." menu in the upper left corner. Click the menu at the top left to start importing the template.

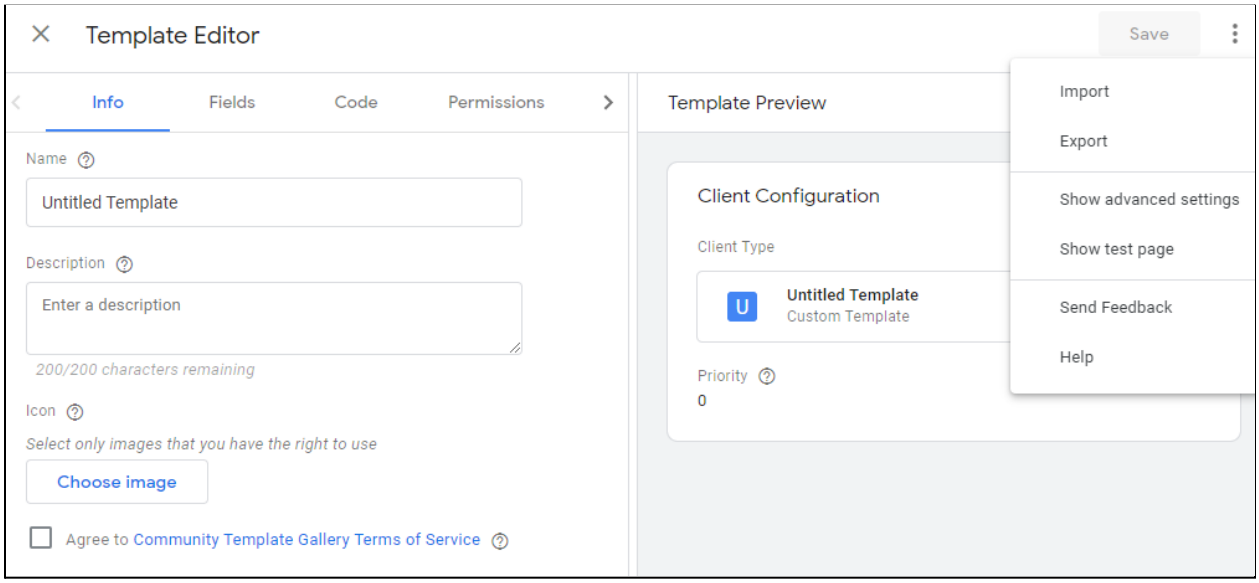


Figure 4: Importing a template from a file

After selecting the import file, all settings are collected and the information, fields, code, etc. are transferred. Afterwards, the template can be saved and used to create new clients in the server container.

Note: This particular client does nothing malicious and is only relevant for the GTM preview when manually submitting test requests from the browser

Explanation: In the course of template development, test data is occasionally generated by making manual requests to the server. Every time a request is manually sent to the tag server from the address bar in the browser, the server also retrieves an icon (favicon.ico). These requests are not always visible in the debugger as an event, but still the client responds to requests for the icon with a symbol: The logo of the Klingon Empire.

This is due to the author's inclination towards Star Trek and his sense of humor. Installing the client in your own ssGTM to have a first look into the template editor is explicitly recommended ;)

In this way, existing templates for clients, tags and variables can be transferred to your own container and used there. As long as there are existing solutions for your own requirements, you do not have to write your own code. Individual requirements, however, usually call for individual solutions. Exactly this will now be demonstrated in the following examples.

Goals / example scenarios

This chapter deals with the basics and the concrete development of templates for tags, clients and variables. The order is based on the typical requirements that arise when setting up server-side tagging with the ssGTM.

The following requirements will be covered, which can not be implemented without custom templates:

Tag: Matomo connection

Matomo, as a first party web analytics, is capable of being operated without cookies and without explicit consent if configured appropriately [31]. An assessment that is shared by many (but not all) data protectionists. For this reason, Matomo is very popular for "substitute data collection" in case of incomplete data collection in Google Analytics due to a lack of tracking consent.

Usually, in addition to the Google Analytics tracking code, another code for Matomo is also implemented in the client.

In the example - limited to page views - the transfer of all received hits to Matomo via a custom tag template will be implemented. The result is a good starting point for your own, complete Matomo connection and shows all essential concepts for tag development.

Client: Use own data format

In the client example, a custom data format is used as the data source for ssGTM. The requests will be received, transferred to the standard event model and, in addition to being passed on to Google Analytics (Universal Analytics), optimized specifically for use in Matomo to support the previously created tag.

Variable: Extract click IDs

The extraction of a click ID from a URL is usually not very difficult. On the server, however, this URL is only available as a complete field in the event model. Extracting individual parameters from this information is not possible with “on-board” tools. In addition, there is now more than one parameter with which Google Ads passes a reference, which makes this particular task even more complex.

Although the problem can be solved with existing Community Gallery templates, the example creates a variable specialized for this specific task.

Data enrichment

A separate chapter deals with the possibilities of retrieving data from APIs or reading and writing information for passing on to tags. Here you will also find sample code for using *Promises* for data enrichment variables and *Firestore* as persistence using the example of a changing salt for hash functions.

Bonus: Tag template for client-side Google Tag Manager

Since templates for the Google Tag Manager are very similar for both variants, the bonus chapter covers a tag template that can be used in the "normal" Tag Manager in the browser to generate tracking requests in exactly the format that is used in the alredy generated ssGTM client template. Thus, a complete tracking from the browser via ssGTM to Facebook, Google Analytics or other services can be set up using this own format.

Scope: the infamous 80%

In order to demonstrate the process of development and at the same time avoid going beyond the scope of this book, each example will make certain sacrifices in comfort, configuration and functionality. The goal is not a ready-to-use template for Matomo or custom request formats, but an understanding of the approach and essential concepts in template development.

For this purpose, usually 80% of the functional scope must be sufficient, which can be produced comparably fast. Using the tag as an example, submitting page views needs to be enough for the sake of demonstration. If events, e-commerce and others are also to be processed, the remains must be implemented independently following the same process.

However, the examples are well suited as a starting point. If you want to feed a service like *Plausible Analytics* [30] with data, you have to compile and send other parameters in a different way, but the concept stays the same.

Anyone who takes the trouble to put the work required for a typical application into their own template should definitely also post the result in the Community Gallery. The hurdles are low and the potential benefit for others who want to solve the same problem with ssGTM is very high. Therefore: Be brave! More encouraging details follow at the end of this book.

Attention: This is not a JavaScript course!

Although the development of the templates from the examples is shown in detail...

Bonus: "Snoop Log Pixel" template for client-side Google Tag Manager

In this bonus chapter, based on the theoretical tracking format for the *Snoop Logg* client on the server, we will lay the foundation for a real-world deployment on a website.

Fortunately, with the knowledge gained about how templates work and how to develop them on the server, this hardly involves any major changes. This is because almost all the principles presented in the previous chapters for developing your own server-side GTM templates also apply in the browser.

Despite all the differences in the purpose of GTM in the browser and on the server, both sides are about tracking. In addition, there are many similarities in the interface for both container types.

The commonalities also include the template editor and everything related to it described in the previous chapters:

- JavaScript in the development sandbox
- APIs and permissions
- Configuration fields

The differences between the two types of templates mainly concern the "living conditions" of both containers. In ssGTM, everything revolves around receiving tracking data, processing it and passing it on. In the ecosystem of a server.

Client-side templates exist in the browser. Here there is access to all the information that is available at "runtime" of a website. This includes DOM properties, dataLayer, cookies and browser-side storage like localStorage etc..

The main task of GTM in the browser in the context of tracking is to collect the information necessary for tracking an event such as a page view. It is usually supported by the dataLayer and provided with information that is useful for tracking.

The set of available APIs therefore differs from the server, which is due to the different task areas and runtime conditions. Overlaps are nevertheless definitely present. Enough, anyway, to dare to create a tag template for the browser.

Step 1: Create a testing environment

Please do not forget: A prerequisite for all of the following steps is a client-side "*Web*" container for Google Tag Manager. Everything that has happened so far in this book has been implemented in a "*Server*" type container.

Therefore, from here on, change the container and either choose an existing container that is embedded on a real website or create a completely new container.

In case of an existing container, a new workspace should be used as a failsafe, where the changes are isolated and reasonably safe from being accidentally published. If a new container is used, it can be "injected" into any existing website using a Chrome extension such as *GTM Helper* [26] to test the tag in the container's preview without actually having to implement it on the site.

Step 2: Create new tag template in Tag Manager

In the web container, there is also a "*Templates*" area in which a new entry is created under "*Tag templates*". The displayed template editor corresponds 1:1 to the one from the server container. As there, everything starts with a name, which is entered as a description on the "*Information*" page.

Since the new tag is intended to send requests in "*Snoop Logg*" format to an ssGTM, "*Snoop Logg Pixel*" is a good name. After that you can go directly to the "*Code*" tab to create and test a first version of the template.

It starts with - this is now almost routine - a list of APIs that will be used in the tag. As with the server templates, a few APIs are sufficient for a first version, which will be added to in the course of building the template.

```
const getUrl = require('getUrl');
const getReferrerUrl = require('getReferrerUrl');
const readTitle = require('readTitle
encodeUriComponent = require('encodeUriComponent');
const sendPixel = require('sendPixel');
```

Since the names of the APIs are speaking, their purpose is easy to read. For a lot of data that is needed to create a tracking hit, a separate API is responsible that allows access to the respective information. This allows very granular determination of what data a tag can read and process in GTM - just like server templates.

For example, from the table of parameters in the section "*The Snoop Logg Format*", the following information is required or processed by the client on the server. This also includes values that were added later in the supplement for Google Analytics.

- lc = "Location" / URL
- tl = page title
- rf = referrer

This data is read in via the first three APIs. The fourth is used to encode the values for use as URL parameters (which has already been used on the server). The last API will send the hit to your own endpoint.

Using the best practices from the chapter of the same name, the first code version uses an *addParam()* titled function to build and send an initial tracking hit from the information (and some constants) obtained through the above APIs.

```
const getUrl = require('getUrl');
const sendPixel = require('sendPixel');
const getTimestampMillis = require('getTimestampMillis');
const encodeUriComponent = require('encodeUriComponent');
const getReferrerUrl = require('getReferrerUrl');
const readTitle = require('readTitle');

var addParam = function(name, value) {
  if (value) return "&" + name + '=' +
  encodeUriComponent(value.toString()); else return "";
};
var url = data.endpointUrl ||
"https://gtm-xxxxx-yyyyy.uc.r.appspot.com/snooplogg";

//Create parameter string and send GET request
if (!url)
  data.gtmOnFailure();
else {
  var en = data.eventName || "PageView";
  var params = "?ev=" + en +
      addParam('tl', readTitle()) +
      addParam('rf', getReferrerUrl()) +
      addParam('lc', getUrl());

  sendPixel(url + params, data.gtmOnSuccess(), data.gtmOnFailure);
}
```

Since there are no settings yet that make the URL of the endpoint and the event name definable via the *data* object, fallback values are added for both variables that allow a test of a PageView on your server endpoint. The fact that there are no settings yet does not interfere with the executability of the code. The reason is explained in step 2 of the chapter on variable templates.

Note: For the test, the default endpoint as fallback (marked red) for the *url* in the code must be replaced with the address of your own server!

If there is no own endpoint, the URL https://httpbin.org/anything can be used as an alternative. This is an endpoint that returns all request data in the response. It was already used in the chapter on server-side tag development.

Before the code can be saved, permissions must be defined again that affect the included APIs. The tag example is given maximum freedom here. Unlike the other templates in this book, however, the permissions assigned with "*All*" or "*Any HTTPS URLs*" are actually suitable for live operation. This is because the complete URLs of the page and the referrer should be used and the hit can be sent to any endpoint (which can be defined later in the settings).

Create tag

After saving the template, a new tag can be created in the "*Tags*" area of the GTM using the template. As a trigger, "*All pages*" is sufficient for test operation.

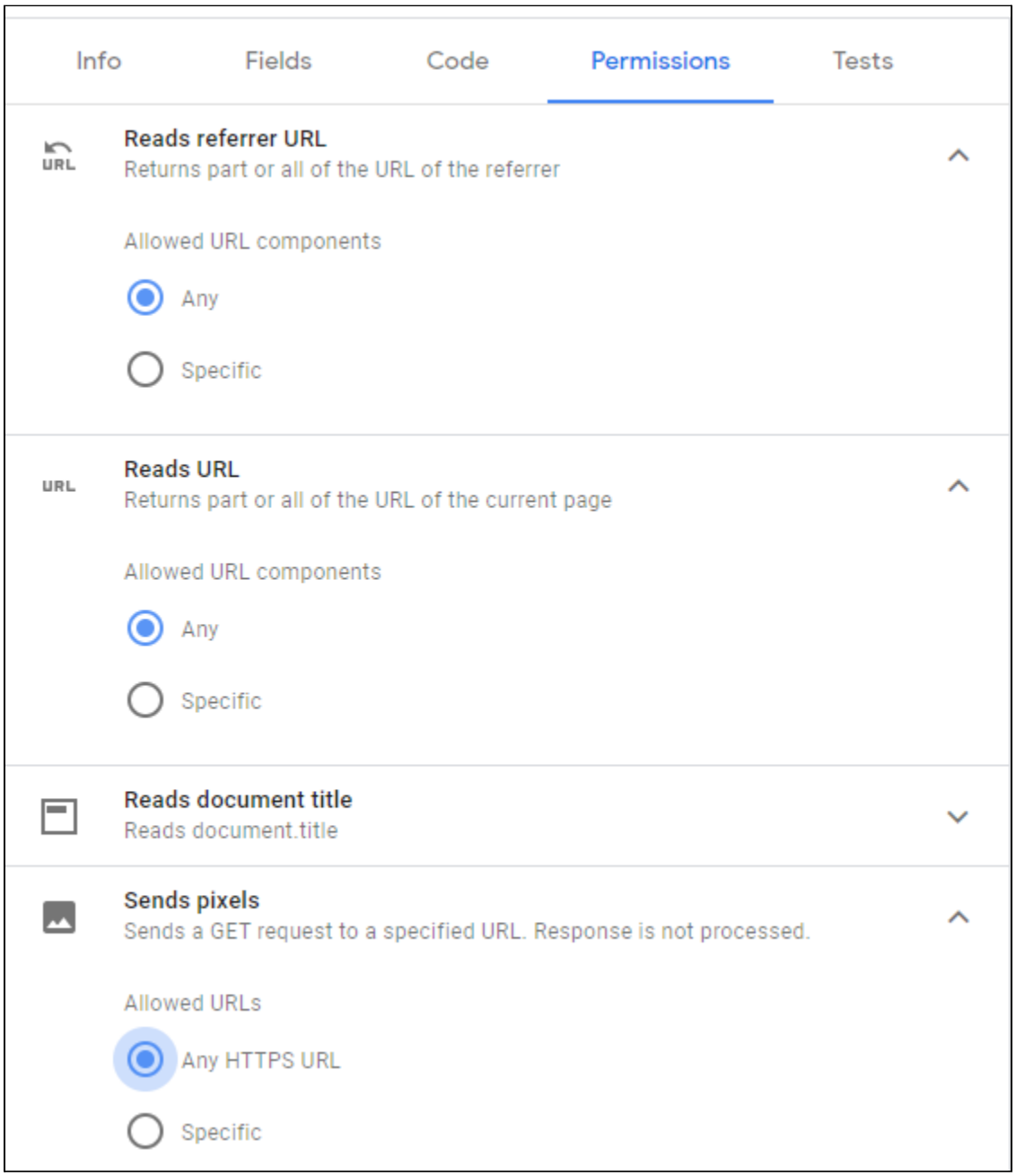


Figure 36: Maximum freedom for the tag APIs

If the Tag Manager is now put into preview, the debugger of the Tag Assistant shows the firing of the new tag in a separate tab at the "*Container Loaded*" event. The feedback of the tag to the GTM should be "*Succeeded*".

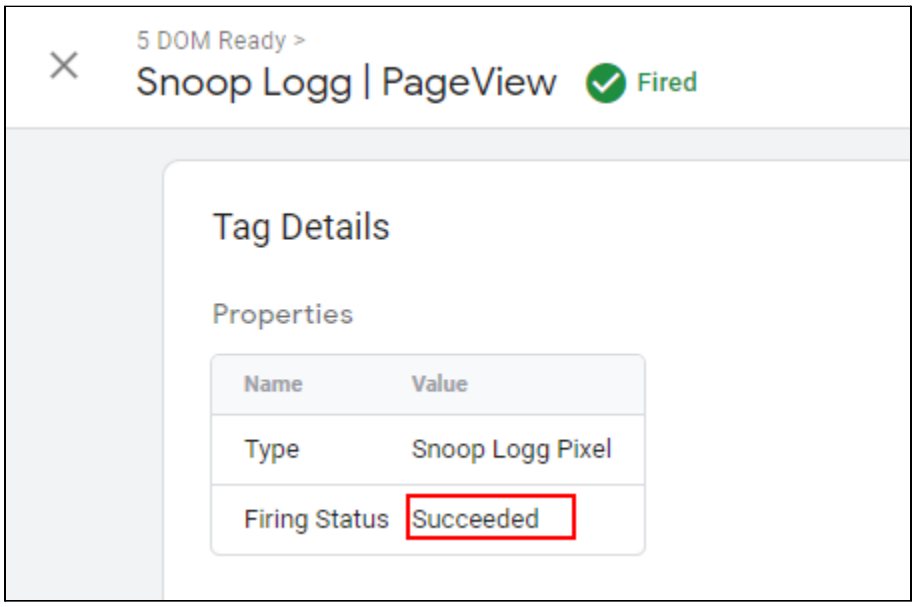


Figure 37: Success message in the debugger

This feedback is based on calling `data.gtmOnSuccess()` on the success callback of the `sendPixel()` function at the end of the code. The counterpart `data.gtmOnFailure()` is called either when something went wrong with the send (by assigning the function on the error callback) or when there is no *url* (which should actually never occur).

The documentation of the client-side APIs [28] also explains this assignment. In addition, the principle of asynchronous processing of all functions relating to the generation of requests and responses is described in the chapter on the server-side tag. In the same way, the outgoing request is sent asynchronously by the tag.

Unfortunately, more feedback is not visible in the debugger. Unless something unexpected happened during runtime. Possible error messages by the tag would be visible in the debugger under *Errors*.

Using the Browser Console

In order to be able to check what happened with the request, the browser console is the better choice for Web-GTM tags. Everything that is logged in the tag template, e.g. using *logToConsole(),* also ends up there. For testing purposes, the following statement can be added anywhere within the template code.

```
require('logToConsole')('logToConsole - messages also end up here.');
```

After saving the template and updating the GTM preview, the message appears in the browser when the console is opened (in e.g. Chrome via CTRL + SHIFT + J).

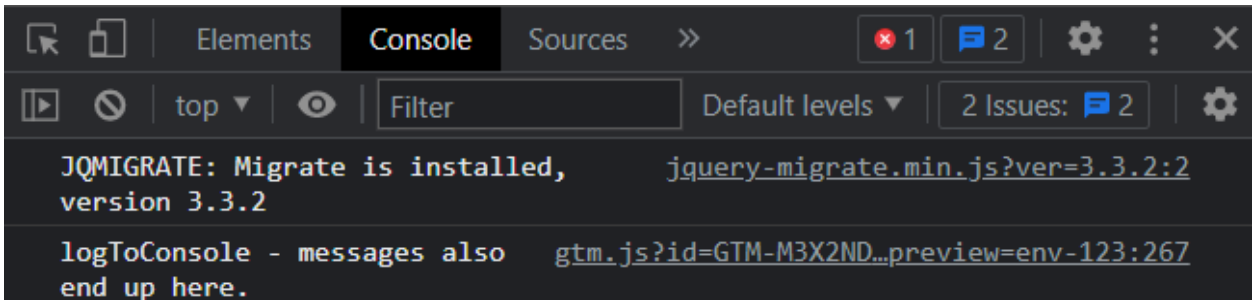


Figure 38: Messages in the browser console

If you also see an error message that affects the specified endpoint for the tag, your own ssGTM may currently not be accessible. In case of a server in test mode, this is due to the fact that it is not currently being operated in the preview and is therefore "asleep".

Regardless of the result, the request must show up on "Network" when switching to and in the list of requests when searching for "*snoop*". The details of the location can be displayed by clicking on it. At the bottom of the "*Request*" you will find a list of the passed parameters.

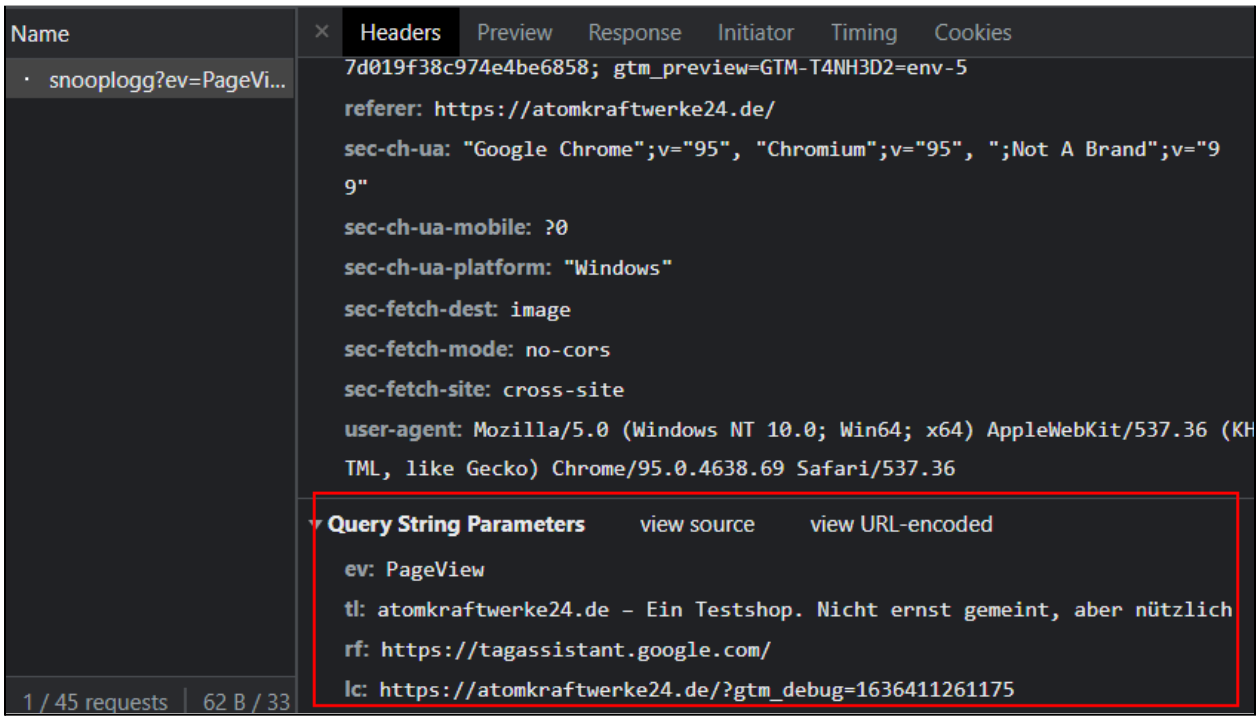


Figure 39: Successfully sent tracking request to the own ssGTM

Also above the list you can see what headers were sent with the request... and what ssGTM answered.

Result at ssGTM

It is much more interesting to see what was done with the request in the preview of the server-side container by the self-created *Snoop Logg* client. The client should have understood the request and converted it into an event. For this event, the data sent by the tag in the browser and translated into the event model is displayed on the Event Data tab, when the server-side container is in preview mode as well.

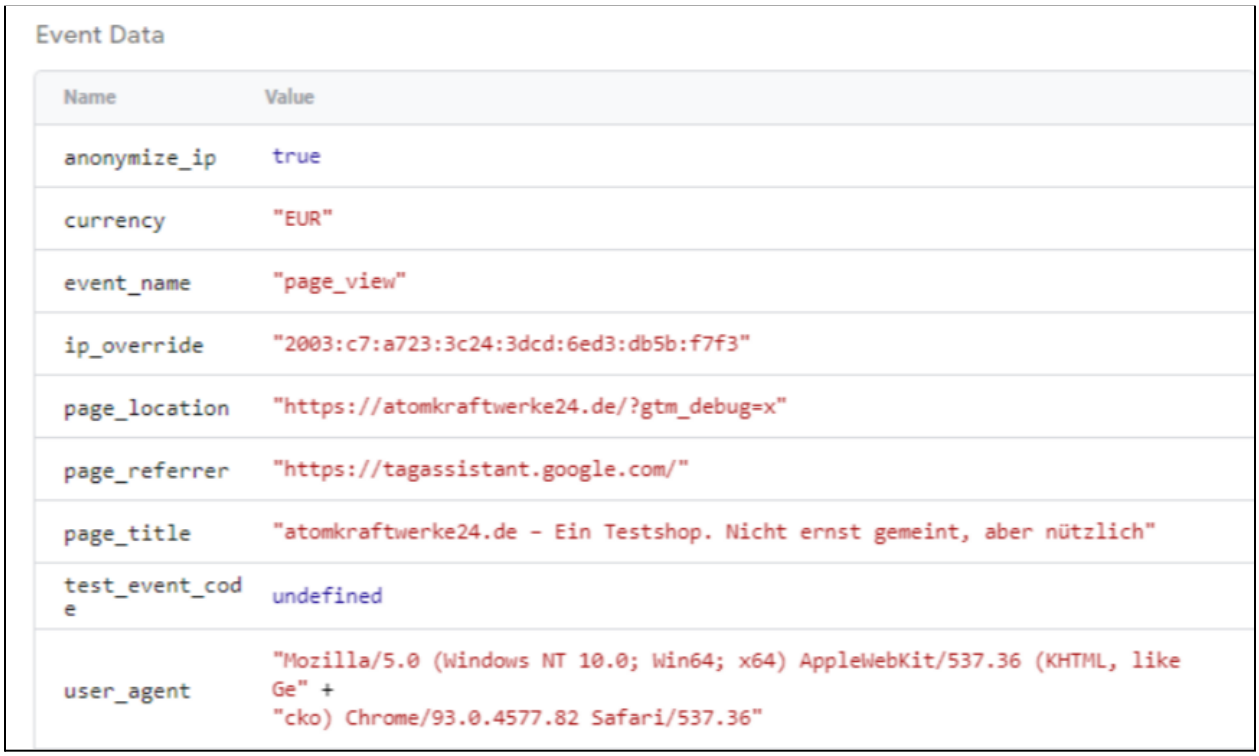


Figure 40: Result at ssGTM of the data sent so far in the event model

For the first time, data was collected on the website in its own format, sent to your own server and processed there. If tags were still triggered on the server for transfer to Facebook or Analytics from the previous chapters, the data also left the ssGTM in a targeted manner.

Congratulations! The whole tracking process has gone through for the first time based on completely self-made templates. This is something to celebrate!

Result when using the test endpoint

If no ssGTM was used for the reception, but the test endpoint at `https://httpbin.org/anything`, a check in the Network Tab of the console is also - conditionally - possible. There you can see both the status code 200 and the details of the

outgoing request. So far there is no difference to the case described above when using an ssGTM.

To view the details of the response, the entry with the call from the list of requests can be double-clicked. In a new tab, the URL is retrieved again and shows the response of the test server. Just as with the test operation of the server-side tag, the parameters are listed separately in the "Result object" under *args*.


```
{
  "args": {
    "ev": "PageView",
    "lc": "https://atomkraftwerke24.de/?gtm_debug=x",
    "rf": "https://tagassistant.google.com/",
    "tl": "atomkraftwerke24.de \u2013 Ein Testshop"
  },
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ima
ge/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
...
}
```


So the dispatch works - now the necessary settings and further data are missing, which should be processed by the tag template.

Step 3: Additional functions and settings

In order to provide the client on the server (yes, that naming *is* really odd) with all the data that is processed there by the current state of the server-side code, the tag template is...