Webtrekk
Drive Your Profit.

Functional Description of the Pixel Plug-in Interface

Pixel Version 3.2.7

Webtrekk
Drive Your Profit.

Contents

Webtrekk
Drive Your Profit.

1 General

The current version of the Webtrekk tracking script (Version 3.2.x) allows various plug-ins to be embedded. All JavaScript functions passed to the Webtrekk pixel are considered as plugins. Such plug-ins can be provided by Webtrekk or developed by you individually.

With the help of plug-ins, third-party tracking scripts can be embedded to fill additional Webtrekk parameters or change data before sending it, for example.

The plug-in scripts give you access to all Webtrekk parameters and functions in the Webtrekk tracking script to make your work easier.

This document describes the process of integrating a plug-in into the Webtrekk pixel environment as well as utilisation of the functionality.

2 Activating the Plug-in

The parameter “executePluginFunction” is used to activate the plug-in functionality. This can either be set in the configuration variable (webtrekkConfig) or as a parameter of the pixel object.

One or more function names can be assigned to the variable “wt.executePluginFunction”, and these are then executed in sequence.

The functions passed on the pixel must be available globally under the object “window”. Multiple plug-ins are separated using a semicolon.

Example for a complete page-specific configuration:

```
var wt = new webtrekkV3();
wt.contentId = "de.startseite";
wt.executePluginFunction = "plugin1;plugin2;plugin3;plugin4;plugin5";
wt.sendinfo();
```

Example for a complete global configuration:

```
var webtrekkConfig = {
        trackId : "111111111111111",
        trackDomain : "track.webtrekk.net",
        domain : "www.website.com",
        cookie : "1",
        contentId : "",
        executePluginFunction : "plugin1;plugin2;plugin3;plugin4;plugin5"
};
```

3 Structure of a Plug-in Function

Your plug-in function is a JavaScript function containing any code that you require. The methods described in chapter 4 should help you when writing the functions.

It is important, however, to ensure that the variable “conf” is passed to your function. This contains important information on the executed pixel request. It will often be the case that your plug-in should only be executed with certain request types, such as page requests, and the “conf” details will enable you to check such information.

```
function pluginX(conf)
{
        …. my code …
}
```

The “conf” object contains the following information.

Parameter	Data Type	Description	Example
instance	webtrekkV3	Contains the actual pixel instance	wt
mode	String	Type of tracking request	“link”, “click”, “page”, “heatmap” or “form”
type	String	Contains information on whether the plug-in was executed before or after a track request	“before“ or “after“
requestCounter	Integer	Contains a number that shows how often this type of request has now been executed	5

4 Parameter Overview

When executing a plug-in, the function in question becomes an internal function. You therefore have the possibility to access all internal functions and variables using “**this**” or “**conf.instance**”. The following table contains a description and explanation of all variables and functions.

Any variables that are not assigned a value are declared with the data type “Boolean” and the standard value “false”.

Property	Data Type	Description	Example
birthday	String	Date of birth of the customer (YYYYMMDD)	19901027
birthdayD	String	Birthday of the customer (DD)	27
birthdayJ	String	Year of birth of the customer (DD)	1990
birthdayM	String	Month of birth of the customer (DD)	10
browserLang	String	Browser language	de
campaignAction	String	Campaign action	“click“ or “view“
campaignId	String	Contains the campaigns and media code	mc%3DNewsletter_2011_08
city	String	City	Berlin

config	Objekt	Contains information for the actual track request, the object is emptied after being sent	config : {
			contentId : "de.Startseite"
			}

contentGroup	Objekt	Contains the page categories	contentGroup : {
			1 : "Startseite"
			}

contentId	String	Page name	de.Startseite
cookie	String	Cookie setting used	“1“ or “3“
cookieDomain	String	Domain in which the cookies should be set	.webtrekk.com
country	String	Country	Germany

couponValue	String	Coupon value without currency	25
currency	String	Currency	EUR

customCampaignParameter	Object	Contains the campaign parameter	customCampaignParameter = {
			1 : "personalized"
			}

customClickParameter	Object	Contains the action parameter	customClickParameter = {
			1 : "action1",
			7 : "action7"
			}

customEcommerceParameter	Object	Contains the product parameter	customEcommerceParameter = {
			1 : "L;32",
			2 : "grün;blau",
			3 : "rechnung"
			}

customParameter	Object	Contains the page parameter	customParameter = {
			1 : "green"
			}

customSessionParameter	Object	Contains the session parameter	customSessionParameter = {
			1 : "logged_in"
			};

customerId	String	Contains the customer ID	372d1a04d003eebc09e1
deactivatePixel	Boolean	Deactivates the entire pixel transmission	true
deactivateRequest	Boolean	Deactivates pixel transmission for the current request	true
delayLinkTrack	Boolean	Activates the delay for link tracking	true
delayLinkTrackTime	Number	Time in milliseconds for the link tracking delay	200

disableOverlayView	Boolean	Deactivates the overlay view or heatmap	true
domain	String	Contains the domain of the page to be tracked	www.website.com
eid	String	Contains the user ID that is determined automatically by the pixel. This value may not be changed.	2132385537700174775
email	String	E-mail address	test@tester.com
emailOptin	String	Receipt of a newsletter	“1“ or “0“
emailRID	String	E-mail RID	123456789
executePluginFunction	String	Contains the name of the plug-in to be executed	plugin1;plugin2;plugin3
firstName	String	First name	Hans
forceHTTPS	String	Ensures the pixel requests are always sent with SSL	1
form	String	Activates form tracking	1
formAnonymous	String	Anonymises all form data	1
formAttribute	String	Used to read the form name	rel
formFieldAttribute	String	Used to read the form field name	id

formFieldDefaultValue	Object	Contains standard values for prefilled form fields	{
			fname: "First name"
			}

formFullContent	String	Details of the form field name, for which all content should be sent	town/city;postcode
formPathAnalysis	Boolean	Activates path analysis for the form focus	true

formSubmit	Boolean	Determines whether a form should be sent	true
formValueAttribute	String	Determines an alternative attribute for reading the form field content	value
gender	String	Gender	“0“, “1“ or “2“
heatmap	String	Activates the heatmap	"1"
heatmapRefpoint	String	Reference point for the heatmap	wt_refpoint
internalSearch	String	Contains the search term for an internal search	ssearch term
isChrome	Boolean	Determines whether a Google Chrome browser is being used	false
isIE	Boolean	Determines whether an Internet Explorer browser is being used	true
isOpera	Boolean	Determines whether an Opera browser is being used	false
isSafari	Boolean	Determines whether a Safari browser is being used	false
linkId	String	Contains the name of the clicked link	Link name
linkTrack	String	Activates link tracking	“link“ or “standard“
linkTrackAttribute	String	Can be used to generate the action name	name
linkTrackDownloads	String	Contains the file extension of the document	doc;pdf;zip
linkTrackIgnorePattern	String	Regular expression that is used to exclude links from link tracking	/^javascript:.+/

linkTrackParams	String	Can be used to generate the action name	teaser_id;page_id
linkTrackPattern	RegExp	Filters an area out of the URL and replaces this with the variable content “inkTrackReplace”	new RegExp(";jsessionid=[a-zA-Z0-9\.]+[#|?|&]?","g")
linkTrackReplace	String	Replaces the item found for the parameter “linkTrackPattern” with the given value	"?"
linktrackOut	Boolean	Determines whether the link is an external link	true
maxRequestLength	Number	Maximum request length	7168
mediaCode	String	Contains the name of the media code	mc
mediaCodeCookie	String	Track media code just once per session or within a certain timeframe	sid or eid
mediaCodeValue	String	Contains the value for the media code that was found	Newsletter_2011_08
noDelayLinkTrackAttribute	String	Attribute for flagging links with a delay	data-wt-delay
optOut	Boolean	Deactivates the entire pixel transmission	true
optoutName	String	Name of the opt-out cookie	webtrekkOptOut
orderId	String	Contains a unique order number	M-12345
orderValue	String	Contains the overall order value	12.99
pixelSampling	String	With this value only every tenth (or respective number) of visitors will be tracked	10

plugins	Array	Contains all plug-ins supported by the browser	["Adobe Acrobat", "Windows Media Player", "Shockwave Flash"]
postalCode	String	Postcode	12345
product	String	Contains the product name	pullover

productCategory	Object	Contains the product parameter	productCategory = {
			1 : "clothing",
			2 : "noname"
			}

productCost	String	Contains the product price	99.90
productQuantity	String	Contains the product quantity	2
productStatus	String	Contains the shopping basket status	“view“, “add“ or “conf“
street	String	Street	Test street
streetNumber	String	House number	12
telefon	String	Phone number	0123456789
trackDomain	String	Contains the track URL	q3.webtrekk.net
trackId	String	Contains the 15-digit Webtrekk customer ID	111111111111111
updateCookie	Boolean	Webtrekk EID will be set to another 6 month duration for each PI	true

urmCategory	Object	Contains the user parameter	urmCategory = {
			1 : "27",
			2 : "single"
			}

version	Integer	Contains the pixel version	321

cookieManager	Constructor	Used to set and read cookies	See chapter 5.6
include	Function	Used to load JavaScript files	See chapter 5.2
loadAsynchron	Function	Used to load JavaScript files with details of a call-back function	See chapter 5.3
registerEvent	Function	Used to register events	See chapter 5.4
unregisterEvent	Function	Used to delete events	See chapter 5.5
sendinfo	Function	Global function for sending track requests	See chapter 0

5 Function Overview

This section describes the internal Webtrekk functions from the pixel, which will help you to develop your own plug-ins.

5.1 Function – sendinfo

The “sendinfo” function is the central method for sending the tracking pixel. When passing the configuration object “c”, only the passed values (in the object) will be sent. As soon as the parameter is no longer passed, the configuration from the tracking pixel is used and sent. With the parameter “p”, a ContentId or LinkId can also be sent. Under “mode” you can define whether the request should be sent as a “click”, “link” or “page” request. If the parameter is not given, “page” is used by default. As soon as a LinkId has been defined, the standard value is “click”. With the last parameter “ep”, a list of custom parameters separated with a semicolon can be passed.

```
/**
 * @param {Object} [c]          Configuration object [optional]
 * @param {String} [p]          ContentId or LinkId [optional]
 * @param {String} [mode]       ”ink“, ”click“ or ”page“ [optional]
 * @param {String} [ep]         Semicolon-separated list of custom parameters [optional]
 */
sendinfo(c, p, mode, ep);
```

5.2 Function – include

The “include” function loads additional JavaScript files.

```
/**
 * @param {string} url    URL of the JavaScript file to be loaded
 * @return true/false
 */
include(url);
```

In the following example plug-in, the file javascript.js file is loaded after the first page request. This takes place asynchronously.

Example for loading a JavaScript file:

```
function plugin1(conf)
{
	if(conf.mode == "page" && conf.type == "after" && conf.requestCounter == 1)
	{
		var url = "http://domain.de/demo/javascript.js";
		this.include(url);
	}
}
```

5.3 Function – loadAsynchron

The function “loadAsynchron” works in the same way as “include”, but this variant allows you to provide more than just the path to the JavaScript file. A variable, function or object must also be defined, which must be located in the file that is to be loaded. This can be used to ensure that the JavaScript file was actually loaded and read/executed successfully by the browser.

Optionally, a call-back function can also be defined. This is executed once the file has been loaded successfully or a standard time of 2000 milliseconds has expired. The function is passed as the first “true” or “false” parameter, which checks whether the JavaScript file was loaded successfully. The second parameter includes the actual pixel instance.

If the standard time of 2000 milliseconds is too low, you can define a different time in milliseconds.

```
/**
 * @param {String} url              URL of the JavaScript file to be loaded
 * @param {String} search           Variable, function or object, which is located
 *                                  in the file to be loaded
 * @param {Function} [callback]     Call-back function, which should be executed
 *                                  subsequently [optional]
 * @param {Integer} [timeout]       Maximum time in milliseconds that should be waited for
 *                                  for the file to load [optional]
 */
loadAsynchron(url, search, callback, timeout);
```

The example plug-in displayed below is used to load a JavaScript file before the first page request. Once the function has found the “demoObject”, the function “demoCallback” will be executed and an “include true” produced. If the file has not loaded within 3000 milliseconds, an “include false” will be produced.

Example for loading a JavaScript file asynchronously:

```
function plugin2(conf)
{
	if(conf.mode == "page" && conf.type == "before" && conf.requestCounter == 1)
	{
		this.demoCallback = function(result, instance)
			{
				if(result)
				{
					alert("include true");
				}
				else
				{
					alert("include false");
				}
			}

		var url = "http://domain.de/demo/javascript.js";
		var search = "demoObject";
		var callback = this.demoCallback;
		var timeout = 3000;
		this.loadAsynchron(url, search, callback, timeout);
	}
}
```

5.4 Function – registerEvent

The “registerEvent” function registers a function for a given event. The first parameter defines the object for which the event should be registered. The second parameter includes the event name. A function is passed as the final parameter, which is executed when the event occurs.

```
/**
 * @param {Object} object                Object with which the event will be registered
 * @param {Object} event                 Event
 * @param {Function|String} function     Function to be executed
 */
registerEvent(object, event, function);
```

The example plug-in displayed below is used to mark each link on the page. The link URL will be passed on as soon as a link is clicked.

Example for registering an event:

```
function plugin3(conf)
{
	if(conf.mode == "page" && conf.type == "before" && conf.requestCounter == 1)
	{
		this.checkLink = function(e)
		{
			var a = document.all ? window.event.srcElement : this;
			alert(a.href);
		}
		var f = this.checkLink;

		for(var i = 0; i < document.links.length; i++)
		{
			this.registerEvent(document.links[i], "mousedown", f);
		}
	}
}
```

5.5 Function – unregisterEvent

The “unregisterEvent” function is the exact opposite of “registerEvent”; it removes a registered function from the given event. The first parameter defines the object from which the event should be removed. The second parameter contains the event. The final parameter defines a function that should be removed.

```
/**
 * @param {Object} object                Object for which the event should be removed
 * @param {Object} event                 Event
 * @param {Function|String} function     Function that should be removed from the object
 */
unregisterEvent(object, event, function);
```

The example plug-in shown below is used to remove the previously set function “checkLink” from each link.

Example for unregistering an event:

```
function plugin4(conf)
{
	if(conf.mode == "page" && conf.type == "before" && conf.requestCounter == 1)
	{
		this.checkLink = function(e)
		{
			var a = document.all ? window.event.srcElement : this;
			alert(a.href);
		 }
		var f = this.checkLink;

		for(var i = 0; i < document.links.length; i++)
		{
			this.registerEvent(document.links[i], "mousedown", f);
			this.unregisterEvent(document.links[i],"mousedown", f);
		}
	}
}
```

5.6 Function – cookieManager

The Cookie Manager is a “Key Value Store” to save strings for later use. The Key Value Store is used to define a key (i.e. a unique key as a string) for saving with any value (also as a string).

Typical uses, among others, are to loop through parameters whose information is not available on all pages and to restrict the runtime of single parameters.

```
/**
 * @param {String} name                  Name of the cookie
 * @param {Number} expires               Runtime in days
 * @param { String } [accessPath]        Path in which the cookie is set [optional]
 *
 * @type Constructor
 */
cookieManager(name, expires, accessPath);
```

5.6.1 Cookie Manager Methods

5.6.1.1 Method - Set

Sets “value” to “key” and saves the cookie; “value” (depending on the mode selected) is then returned.

The optional third parameter defines a runtime in seconds for the set value. If no value is entered here, the set value remains valid for a year.

The optional fourth parameter, “mode”, can be set to “first” or “last” (default). If “first” is set, the transmitted value is only set at the first call during runtime. All subsequent calls return the value set in the cookie. If “last” is set, the last transmitted value is always saved and returned.

```
/**
 * @param {String} fieldname      Name of the key within the cookie
 * @param {String} fieldval       Value for the given key
 * @param {Number} duration       Duration in seconds
 * @param {String} [mode]         “first” or “last”
 *
 * @returns {String|Boolean}      value or false
 *
 */
set(fieldname, fieldval, duration, mode);
```

5.6.1.2 Method - Get

Returns the value saved in “key”.

```
/**
 * @param {Object}  fieldname     Key be searched for in cookie
 *
 * @return {String}               Value
 */
get(fieldname);
```

5.6.1.3 Method - GetSize

Returns the number of characters saved in the cookie.

A cookie can save around 4 KB of data. If there is not enough memory for a certain application, this method can be used to query the length of the string saved in the cookie. If the size of 4 KB is exceeded, the application creates another cookie to save the data.

```
/**
 * @returns {Number}     Number of characters in the complete cookie
 */
getSize();
```

5.6.1.4 Method - Prepare

This function is structured in a similar way to the method “set”. It is used to hold information for the write operation that changes regularly (such as the visit duration, which is set continuously). The method “write” must be used to write the actual value in the cookie.

This procedure ensures more efficient data processing, as the number of write operations is minimised.

```
/**
 * @param {String} fieldname       Name of the key within the cookie
 * @param {String} fieldval        Value for the given key
 * @param {Number} duration        Duration in seconds
 * @param {String} [mode]          “first” or “last”
 *
 * @returns {String|Boolean}       value or false
*/
prepare(fieldname, fieldval, duration, mode)
```

5.6.1.5 Method - Write

With the help of this method, the values previously set are saved in the cookie.

At the same time, the runtimes of the values saved in it are checked. If the runtime has been exceeded, the corresponding values are deleted from the cookie. If runtimes of all values have been exceeded, the cookie is deleted.

```
write();
```

5.6.1.6 Method - Remove

Deletes a cookie. Data deletion is permanent and cannot be reversed!

```
/**
 * @returns {Boolean}    whereby true means that the cookie was read out again
 *                       and could not therefore be deleted
 */
remove();
```

5.6.2 Cookie Manager Example

Let us assume that additional functions are to be offered for free for one month when registering on our website, and following this time a charge will be made. The “free premium” status is passed during this time.

Solution:

On registering, this status is set by the Cookie Manager for 30 days. The value is then sent as a page or session parameter with every request from each page.

The following code is built into the registration confirmation page:

```
var wt = new webtrekkV3();
var myCookie = new wt.cookieManager("registration", 30);
wt.customSessionParameter = {
    1 :  myCookie.set('status', 'free premium', 2592000, 'first');
};
wt.sendinfo();
```

The following code is built into all other pages:

```
var wt = new webtrekkV3();
var myCookie = new wt.cookieManager("registration",30);
wt.customSessionParameter = {
    1 :  myCookie.get('status');
};
wt.sendinfo();
```

In this example, the “free premium” value for the “status” key is set with a validity of 2,592,000 seconds (= 30 days). In the frontend, we have set a session parameter of 1 for this status and written the value to it. On expiry, calling myCookie.get('status'); returns an empty string and the session parameter is no longer transmitted.

We can now filter out session parameter 1 based on the “free premium” value in the frontend to receive the number of users with this status.

5.7 Changing Internal Variables

The method “this” can be used in the plug-in to access all variables previously defined on the page. “this.config” can be used to access all variables, which are relevant for the current track request.

The following example shows how to extend the “contented” to include the browser language prior to each request.

Example for changing the “contentId“:

```
function plugin5(conf)
{
        if(conf.type == "before")
        {
                var browserLang = "";
                if (typeof(navigator.language) == "string")
                {
                        browserLang = navigator.language.substring(0,2);
                }
                else if (typeof(navigator.userLanguage) == "string")
                {
                        browserLang = navigator.userLanguage.substring(0,2);
                }
                this.config.contentId = browserLang + "." + this.config.contentId;
        }
}
```

6 Coding Rules

A couple of coding rules have been defined in the following chapters, which you should adhere to when developing a Webtrekk pixel plug-in. Once the plug-in has been developed, Webtrekk will carry out a final check and approval of the functionalities.

6.1 Prohibited Function Calls

The plug-in or loaded file may not include any of the following call types:

- document.write, document.writeln, document.open, document.close

6.2 Browsers and Operating Systems

The plug-in should be supported by the following browsers and operating systems. All possible constellations should be tested and supported.

- Internet Explorer 6 – 9, Firefox, Google Chrome, Safari, Opera
- Windows XP, Windows Vista, Windows 7, Mac OS, Linux

6.3 Loading a File

It should not take longer than 2000 milliseconds to load a JavaScript file, and the process should not slow down the loading of the website page.

6.4 Track Request

The sending of a Webtrekk pixel request should not be delayed by more than 500 milliseconds. It should be noted that track requests from third-party providers should not be sent until after the Webtrekk track requests have been sent.

With regard to plug-ins that execute an action, these should not delay the sending of a Webtrekk pixel request.

7 Contact

Please don't hesitate to contact us with your questions about the plug-ins:

Webtrekk GmbH
Hannoversche Straße 19
10115 Berlin

phone +49 (0)30 - 755 415 - 0
fax 030 - 755 415 - 100

support@webtrekk.com
http://www.webtrekk.com