


[Blog](#) [Guides](#) [Snippets](#) [Contact](#) [About](#)

# Integrate Facebook Pixel and Drupal Webform via Google Tag Manager

APR 17, 2021 DRUPAL

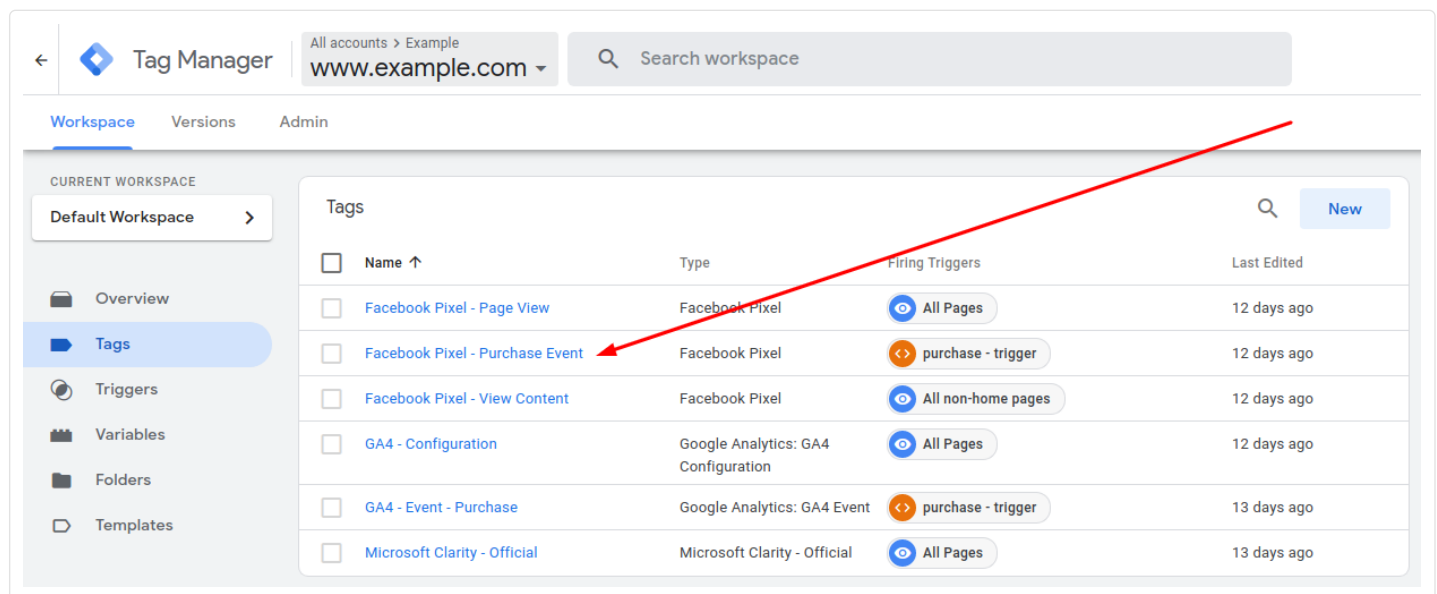
In my previous article, I explained [how to integrate the Facebook Pixel with a Drupal webform](#). I used the [Simple Facebook Pixel](#) module to send the purchase event to Facebook. In this article, we are going to see how to send the purchase event by using Google Tag Manager (GTM).

If you don't already have the Google Tag Manager module installed, you can install it with Composer:

```
composer require drupal/google_tag
```

To use the module, sign up for [GTM](#), get your Container ID, and configure the module. You can find the instructions for module configuration [here](#).

Once we have the GTM module configured on our website, we can create a new tag in Google Tag Manager.



But before we do that, we have to create a data layer, which will contain all information about the purchase. The data layer has to be injected into the HTML after the user makes a purchase.

To inject the data layer we can use the `hook_page_attachments()` like this:

```
use Drupal\Component\Serialization\Json;

function MY_MODULE_page_attachments(array &$page) {
  $data_layer_service = \Drupal::service('MY_MODULE.data_layer_service');
  $data = $data_layer_service->getData();

  if (!empty($data)) {
    $page['#attached']['html_head'][] = [
      '#tag' => 'script',
      '#value' => "window.dataLayer = window.dataLayer || []; dataLayer.push({'event': 'purchase', 'ecommerce': " . Json::encode($data) .
    ],
    'MY_MODULE_data_layer',
  ];
}
```

```

    ];
  }
}

```

Now let's create the **data\_layer\_service** service that acts as a middleman between the *hook\_page\_attachments()* and a Webform handler. This is a simple service that uses **PrivateTempStore** to make temporary data available across requests.

We need the *MY\_MODULE.services.yml* file:

```

services:
  MY_MODULE.data_layer_service:
    class: Drupal\MY_MODULE\DataLayerService
    arguments: [ '@tempstore.private' ]

```

and the *src/DataLayerService.php* file:

```

<?php

namespace Drupal\MY_MODULE;

use Drupal\Core\TempStore\PrivateTempStoreFactory;

class DataLayerService {

  /**
   * The private temp store.
   *
   * @var \Drupal\Core\TempStore\PrivateTempStoreFactory
   */
  protected $privateTempStore;

  /**
   * DataLayerService constructor.
   *
   * @param \Drupal\Core\TempStore\PrivateTempStoreFactory $private_temp_store
   *   The private temp store.
   */
  public function __construct(PrivateTempStoreFactory $private_temp_store) {
    $this->privateTempStore = $private_temp_store->get('MY_MODULE');
  }

  /**
   * Adds data about event.
   *
   * @param array $data
   *   The event data.
   */
  public function addData(array $data) {
    $this->privateTempStore->set('data', $data);
  }

  /**
   * Gets data about event.
   */
  public function getData() {
    $data = $this->privateTempStore->get('data');
    $this->privateTempStore->delete('data');
    return $data;
  }
}

```

Finally, we can create the Webform handler (placed inside *Plugin/WebformHandler* directory) that will use the Data layer service to store information about the purchase.

Go back to Google Tag Manager and define new custom variables for all **\$data** array keys.

&lt;?php

```

namespace Drupal\MY_MODULE\Plugin\WebformHandler;

use Drupal\commerce_product\Entity\ProductInterface;
use Drupal\webform\Plugin\WebformHandlerBase;
use Drupal\webform\WebformSubmissionInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Inserts data layer on a webform submission.
 */
/**
 * @WebformHandler(
 *   id = "data_layer",
 *   label = @Translation("Data layer"),
 *   category = @Translation("Action"),
 *   description = @Translation("Inserts data layer on a webform submission."),
 *   cardinality = \Drupal\webform\Plugin\WebformHandlerInterface::CARDINALITY_UNLIMITED,
 *   results = \Drupal\webform\Plugin\WebformHandlerInterface::RESULTS_PROCESSED,
 *   submission = \Drupal\webform\Plugin\WebformHandlerInterface::SUBMISSION_REQUIRED,
 * )
 */
class DataLayerWebformHandler extends WebformHandlerBase {

  /**
   * The route match.
   */
  /**
   * @var \Drupal\Core\Routing\RouteMatchInterface
   */
  protected $routeMatch;

  /**
   * The data layer service.
   */
  /**
   * @var \Drupal\MY_MODULE\DataLayerService
   */
  protected $dataLayerService;

  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container, array $configuration, $plugin_id, $plugin_definition) {
    $instance = parent::create($container, $configuration, $plugin_id, $plugin_definition);
    $instance->routeMatch = $container->get('current_route_match');
    $instance->dataLayerService = $container->get('MY_MODULE.data_layer_service');
    return $instance;
  }

  /**
   * {@inheritdoc}
   */
  public function postSave(WebformSubmissionInterface $webform_submission, $update = TRUE) {
    /** @var \Drupal\commerce_product\Entity\ProductInterface $product */
    $product = $this->routeMatch->getParameter('commerce_product');

    if ($product instanceof ProductInterface) {
      $submission_data = $webform_submission->getData();

      /** @var \Drupal\commerce_product\Entity\ProductVariationInterface $product_variation */
      $product_variation = $product->getDefaultVariation();

      $unit_price = (float) $product_variation->getPrice()->getNumber();
      $quantity = (int) $submission_data['quantity'];
      $total_price = $unit_price * $quantity;

      $skus[] = $product_variation->getSku();
    }
  }
}

```

```

$contents[] = [
  'id' => $product_variation->getSku(),
  'quantity' => $quantity,
  'item_price' => $product_variation->getPrice()->getNumber(),
];

$data = [
  'num_items' => $quantity,
  'content_type' => 'product',
  'value' => $total_price,
  'currency' => $product_variation->getPrice()->getCurrencyCode(),
  'content_ids' => $skus,
  'contents' => $contents,
];

$this->dataLayerService->addData($data);
}
}
}

```

The screenshot shows the Google Tag Manager interface. On the left, the 'Variables' menu item is highlighted. In the main area, there are two sections: 'Built-In Variables' and 'User-Defined Variables'. The 'Built-In Variables' section lists various variables like 'Click Text', 'Event', 'Form Element', 'Form ID', 'Page Hostname', 'Page Path', 'Page URL', and 'Referrer'. The 'User-Defined Variables' section lists custom variables like 'purchase - content\_ids', 'purchase - content\_type', 'purchase - contents', 'purchase - currency', 'purchase - num\_items', and 'purchase - value'. Red arrows indicate the flow from the 'Variables' menu to the 'Form Element' variable in the 'Built-In Variables' section, and then to the 'User-Defined Variables' section, specifically pointing to the 'purchase - content\_ids' variable.

Name	Type
Click Text	Auto-Event Variable
Event	Custom Event
Form Element	Data Layer Variable
Form ID	Data Layer Variable
Page Hostname	URL
Page Path	URL
Page URL	URL
Referrer	HTTP Referrer

Name	Type	Last Edited
purchase - content_ids	Data Layer Variable	12 days ago
purchase - content_type	Data Layer Variable	12 days ago
purchase - contents	Data Layer Variable	12 days ago
purchase - currency	Data Layer Variable	13 days ago
purchase - num_items	Data Layer Variable	12 days ago
purchase - value	Data Layer Variable	13 days ago

We have the following Data layer variables:



*ecommerce.num\_items*  
*ecommerce.content\_type*  
*ecommerce.value*  
*ecommerce.currency*  
*ecommerce.content\_ids*  
*ecommerce.contents*

An example of how a variable looks like in GTM:


× purchase - contents 📁

Variable Configuration


Variable Type

 Data Layer Variable 

Data Layer Variable Name ?

ecommerce.contents 

Data Layer Version

Version 2 

☐ Set Default Value



> Format Value ?

The last step is to create a trigger and a new tag. The trigger fires on the purchase event:

× purchase - trigger 📁

Trigger Configuration

Trigger Type

 Custom Event 

Event name

purchase ☐ Use regex matching

This trigger fires on


☒ All Custom Events ☐ Some Custom Events


And here's the purchase event configuration in GTM:

×
Facebook Pixel - Purchase Event

### Tag Configuration

Tag Type


**Facebook Pixel**  
facebookincubator
 GALLERY


**Tag permissions**
4 permissions >

Facebook Pixel ID(s)  
xxxxxxxxxxxxxxxx

Event Name  
Standard  
Purchase

Consent Granted (GDPR) ?  
True

Data Processing Options

Data Processing Options force this Facebook event to comply to regional regulations with regard to the processing and selling of user data. [Read this](#) for more information about how to configure this section.


Object Properties

Load Properties From Variable ?  
False

Property Name	Property Value
num_items	{{purchase - num_items}}
value	{{purchase - value}}
currency	{{purchase - currency}}
content_ids	{{purchase - content_ids}}
contents	{{purchase - contents}}
content_type	{{purchase - content_type}}

Triggering

Firing Triggers


**purchase - trigger**  
Custom Event

And that's it. It's not an easier way than what I've shown you the last time, but this approach empowers the marketing team. They can use the same data layer to send purchase events to some other analytics service, like for example to Google Analytics without having to ask you to code something.

[#drupal](#) [#facebook pixel](#) [#google tag manager](#) [#ecommerce](#)

## Further reading

- [Integrate Facebook Pixel and Drupal Webform](#)
- [Simplify Drupal Commerce 2.x checkout by removing "Login or continue as guest" pane](#)
- [Out of Stock feature in Drupal Commerce 2.x](#)

Copyright © 2017-2023 by Goran Nikolovski. All rights reserved. [Top of page](#).

[Privacy Policy](#) | [Terms of Service](#) | [Article Tags](#) | [Search](#)

