Google Analytics allow tracking of shopping activity on e-commerce websites. To help you set up this functionality in Kentico, the system provides methods for macros, transformations and the API that return JSON data for products and orders in a suitable format. For more information, see Integrating Google Analytics Enhanced Ecommerce.

By default, the system's methods do not set all of the optional data fields that can be processed by Google Analytics. If you wish to extend or completely change how the system creates e-commerce data for the purposes of Google Analytics, use the following customization approach:

1. Open your Kentico project in Visual Studio.
2. Create a custom class that inherits from one of the Google Analytics data helper classes.

   Class location

   We recommend adding custom classes as part of a new assembly (Class library project) in your Kentico solution. You need to add the appropriate references to both the assembly and the main Kentico web project.

3. Override the methods of the helper class according to your custom requirements.
4. Register your helper implementation using the **RegisterCustomHelper** assembly attribute.

When you reload the website, the system creates e-commerce JSON data using the custom implementations that you registered.

Google Analytics data helper classes

You can find the following helper classes in the **CMS.Ecommerce** namespace of the Kentico API.

GtmProductHelper

Provides the following methods that you can override:

- **MapSKUInternal** – creates data for a specified product (*SKUInfo*).
  - Must return a *GtmData* object containing the required keys (JSON field names) and values.
  - Customize this method to change the output of the *GetGtmProductJson* macro and transformation method, and the data of products returned by the *GetGtmShoppingCartItemsJson* macro method.
- **MapShoppingCartItemsInternal** – creates data for a collection of products representing shopping cart content (*IEnumerable<ShoppingCartItemInfo>*).
  - Must return an *IEnumerable* collection of *GtmData* objects, each containing the required keys (JSON field names) and values.
  - Customize this method to change the output of the *GetGtmShoppingCartItemsJson* macro method.

GtmOrderHelper

Creates data for completed orders (purchases). Customize the methods in this class to change the output of the *GetGtmPurchaseJson* macro method.

Provides the following methods that you can override:

- **MapPurchaseInternal** – creates the overall purchase data for a specified order (*OrderInfo*).
  - Must return a *GtmData* object containing the required keys (JSON field names) and values.
  - By default calls *MapOrderInternal* to create the data of the *actionField* field, and *MapOrderItemsInternal* to create the data of the *products* field.
- **MapOrderInternal** – creates data representing a summary of the specified order (*OrderInfo*), including values such as the total revenue, tax, shipping, etc.
  - Must return a *GtmData* object containing the required keys (JSON field names) and values.
- **MapOrderItemsInternal** – creates data representing all products within the specified order (*OrderInfo*).
  - Must return an *IEnumerable* collection of *GtmData* objects, each containing the required keys (JSON field names) and values.

GtmDataHelper

Performs general serialization of *GtmData* objects into JSON strings. Affects the final output of methods from both the *GtmProductHelper* and *GtmOrderHelper* classes.

You can override the **SerializeToJsonInternal** method – one overload for individual objects (*GtmData* parameter) and another for collections of objects (*IEnumerable<GtmData>* parameter).

Customize the *GtmDataHelper* if you wish to replace or modify the default JSON serialization, which utilizes the *JsonConvert.SerializeObject* method provided by the *Newtonsoft.Json* library.

Creating conditional customizations

If you wish to return different JSON data for specific scenarios, you can use the **purpose** parameter, which is available for all methods of the Google Analytics data helper classes.

When calling the corresponding macro or transformation methods on your website, set the optional *purpose* parameter. For example:

```
{% GetGtmProductJson(SKUID, "", "searchImpressions") %}
```

You can then evaluate the *purpose* parameter within the method overrides in your custom helper classes, and branch your code according to the specified purpose.

Example – Adding coupon data for orders

The following example demonstrates how to customize the Google Analytics JSON data that the system creates for purchases (completed orders). The customization adds the **coupon** field to the *actionField* object in the purchase data, and sets the value to a comma-separated string containing all coupon codes applied to the order.

Start by preparing a separate project for custom classes in your Kentico solution:

1. Open your Kentico solution in Visual Studio.
2. Create a new *Class Library* project in the Kentico solution (or reuse an existing custom project).
3. Add references to the required Kentico libraries (DLLs) for the new project:
   a. Right-click the project and select **Add -> Reference**.
   b. Select the **Browse** tab of the **Reference manager** dialog, click **Browse** and navigate to the ***Lib*** folder of your Kentico web project.
   c. Add references to the following libraries (and any others that you may need in your custom code):
      - **CMS.Base.dll**
      - **CMS.Core.dll**
      - **CMS.DataEngine.dll**
      - **CMS.Ecommerce.dll**
4. Reference the custom project from the Kentico web project *(CMSApp* or *CMS)*.
5. Edit the custom project's **AssemblyInfo.cs** file (in the *Properties* folder).
6. Add the **AssemblyDiscoverable** assembly attribute:

   ```
   using CMS;

   [assembly:AssemblyDiscoverable]
   ```

Continue by creating a custom implementation of the *GtmOrderHelper* class:

1. Add a new class under the custom project.
2. Make the class inherit from **GtmOrderHelper**.

3. Register the class using the **RegisterCustomHelper** assembly attribute.
4. Override the class's **MapOrderInternal** method and return a **GtmData** object containing data according to your custom requirements:

```
using System;
using System.Collections.Generic;
using System.Linq;

using CMS;
using CMS.Ecommerce;

// Registers the CustomGtmOrderHelper class to replace the default GtmOrderHelper
[assembly: RegisterCustomHelper(typeof(CustomGtmOrderHelper))]

public class CustomGtmOrderHelper : GtmOrderHelper
{
    /// <summary>
    /// Maps the specified OrderInfo object to a GtmData object, containing data
suitable for Google Analytics.
    /// The GtmData object is then serialized into a JSON string, which is
returned by the appropriate macro or API method.
    /// </summary>
    protected override GtmData MapOrderInternal(OrderInfo order, object
additionalData = null, string purpose = null)
    {
        if (order == null)
        {
            throw new ArgumentNullException(nameof(order));
        }

        // Adds the default order data
        var customOrderData = new GtmData();
        customOrderData.Add("id", order.OrderID);
        customOrderData.Add("revenue", order.OrderGrandTotal);
        customOrderData.Add("tax", order.OrderTotalTax);
        customOrderData.Add("shipping", order.OrderTotalShipping);

        // Parses the order's coupon code data
        CouponCodeCollection orderCouponCodes = CouponCodeCollection.Deserialize
(order.OrderCouponCodes);

        if (orderCouponCodes != null)
        {
            // Gets all coupon codes applied to the order
            IEnumerable<ICouponCode> appliedCouponCodes = orderCouponCodes.
AllAppliedCodes;

            // Creates a comma separated string containing the coupon codes
            string couponString = String.Join(",", appliedCouponCodes.Select
(coupon => coupon.Code));

            // Adds the coupon field to the order data
            customOrderData.Add("coupon", couponString);
        }

        // Merges the custom data with any additional JSON data provided by the
additionalData parameter
        customOrderData = GtmPropertiesMerger.Merge(customOrderData,
additionalData);

        return customOrderData;
    }
}
```

Calling the base method

The example above fully overrides the default implementation of the *MapOrderInternal* method. If you only wish to extend the default functionality, you can simplify the customization by calling the base method to create the initial *GtmData*.

```
GtmData customOrderData = base.MapOrderInternal(order, additionalData);
customOrderData.Add("customField", "customValue");
```

5. Save all changes and **Build** the custom project.

The customization ensures that the system sets the **coupon** field when creating the purchase JSON data for orders that contain at least one coupon code. For example, the **GetGtmPurchaseJson** macro method could now produce the following JSON output:

```
{
  "actionField": {
    "coupon": "CouponCode1,CouponCode2",
    "id": 123,
    "revenue": 46.99,
    "shipping": 3,
    "tax": 4
  },
  ...
```