By using the Google Analytics Enhanced Ecommerce features, you can measure and analyze shopping activity on your website (product impressions, purchases, etc.). Kentico provides an API that helps get the data of products and purchases in a format suitable for logging via Google Analytics.

Visit the Google Analytics website to set up a Google Analytics account for the website that you wish to monitor.

> **Portal engine websites**
>
> The information on this page applies to e-commerce sites developed using ASP.NET MVC.
>
> If you are developing your on-line store using the portal engine, see: Integrating Google Analytics Enhanced Ecommerce

Adding Google Tag Manager

Before you can start measuring e-commerce activities, you need to add Google Tag Manager to the pages of your website. Copy the snippets from the Google Tag Manager Quick Start Guide into the appropriate sections of your MVC website's main layout (view).

You can then start collecting your site's e-commerce data and tracking events using Google Analytics. See the sections below for more information about setting up individual types of tracking.

Collecting e-commerce data

We recommend using a data layer to collect and send data to Google Analytics. For detailed information, see the Enhanced Ecommerce (UA) Developer Guide.

The Kentico API provides methods that allow you to get suitable JSON data for product and order objects, which you can push into the data layer. The following helper classes are available in the **CMS.Ecommerce** namespace of the standard Kentico API (provided by the *Kentico.Libraries* integration package).

GtmProductHelper class:

- **MapSku** – returns a *GtmData* object containing the data of a specified product (*SkuInfo* object). Provides the following data fields by default: *id* (the product's *SKU* value), *name*, *price* and *brand*
- **MapShoppingCartItems** – returns an *IEnumerable<GtmData>* collection containing the data of specified products in a shopping cart (*IEnumerable<ShoppingCartItemInfo>* collection). By default, provides the same data fields as the *MapSku* method and also adds the unit *quantity* for each product.

GtmOrderHelper class:

- **MapPurchase** – returns a *GtmData* object containing the overall purchase data for a specified order (*OrderInfo* object). Provides the following data by default:
  - *actionField* object representing the overall purchase, with the following fields:
    - *id* (the integer ID of the Kentico order object)
    - *revenue* (the order's *GrandTotal* value, i.e. the final price including discounts, shipping, tax, and reductions from applied gift cards)
    - *shipping*
    - *tax*
  - *products* object containing all products within the order (the product data format is the same as provided by the *GtmProductHelper.MapShoppingCartItems* method)
- **MapOrder** – returns a *GtmData* object containing a summary of the specified order (*OrderInfo* object). Provides the data fields described for the *actionField* object within the data returned by the *MapPurchase* method.
- **MapOrderItems** – returns an *IEnumerable<GtmData>* collection containing the data of all products within the specified order (*OrderInfo* object). Provides the same data as the *products* object within the data returned by the *MapPurchase* method.

GtmDataHelper class:

- **SerializeToJson** – serializes the *GtmData* objects returned by the *GtmProductHelper* or *GtmOrderHelper* methods into JSON strings. Provides one overload for individual objects (*GtmData* parameter) and another for collections of objects (*IEnumerable<GtmData>* parameter).

Adding custom data

When calling the methods of the *GtmProductHelper* and *GtmOrderHelper* classes, you can add custom data fields via the optional **additionalData** parameter. The methods merge the fields and values of the custom object into the returned *GtmData* objects.

Example

```
GtmProductHelper.MapSKU(sku, new { category = "Apparel" })
```

Use the listed methods to create a Google Analytics implementation that suits the purposes of your website. Alternatively, you can compose the required JSON data manually using your own custom code. The following sections contain best practices for various types of e-commerce tracking on Kentico MVC sites:

Product impressions

To log impressions of products displayed to visitors (for example on product listing pages):

1. Set up a view model class representing individual items in the product list:
   a. Add a property of the *GtmData* type, which will store the product's Google Analytics data.
   b. Get the product's data by calling the **GtmProductHelper.MapSKU** method, and assign it to the new property.

```
using CMS.Ecommerce;
using Kentico.Ecommerce;

public class ProductListItemViewModel
{
    public readonly GtmData GtmSKUJson;
        ...

    // Constructor of the view model class representing items in product
lists
        public ProductListItemViewModel(SKUTreeNode productPage,
ProductPrice priceDetail, string publicStatusName)
    {
        // Creates the Google Tag Manager data for the given product and
assigns it to a property
        GtmSKUJson = GtmProductHelper.MapSKU(productPage.SKU);
        ...
    }
}
```

2. Set up another view model class representing the overall product list:

a. Add a property storing a collection of the listed products (represented by your list item view model class).
b. Add a *string* property, which will store the JSON data of all listed products.
c. Create the product list JSON string by calling the **GtmDataHelper.SerializeToJson** method. Prepare an *IEnumerable* collection containing the *GtmData* values of individual products as the method's parameter. Assign the resulting string into the new property.

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

using CMS.Ecommerce;

public class ProductListViewModel
{
    public IEnumerable<ProductListItemViewModel> Products;
    public readonly string GtmProductsJsonString;

    // Constructor of the view model class representing the overall product
list
    public ProductListViewModel(IEnumerable<ProductListItemViewModel>
products)
    {
        Products = products;
        // Creates the Google Tag Manager JSON data for the product list
and assigns it to a property
        GtmProductsJsonString = GtmDataHelper.SerializeToJson(Products.
Select(product => product.GtmSKUJson));
    }
}
```

3. Adjust your controller actions that display products to work with the view models. For example:

```
public ActionResult Index()
{
    // Gets products of the product page type (via the generated page type code)
    List<LearningProductType> products = LearningProductTypeProvider.
GetLearningProductTypes()
        .LatestVersion(false)
        .Published(true)
        .OnSite(siteName)
        .Culture("en-US")
        .CombineWithDefaultCulture()
        .WhereTrue("SKUEnabled")
        .OrderByDescending("SKUInStoreFrom")
        .ToList();

    // Prepares the view model for the listing page
    var model = new ProductListViewModel(products.Select(
        product => new ProductListItemViewModel(
            product,
            pricingService.CalculatePrice(product.SKU, shoppingService.
GetCurrentShoppingCart()),
            product.Product.PublicStatus?.PublicStatusDisplayName))
        );

    // Displays the action's view with an initialized view model
    return View(model);
}
```

4. Edit the view representing your product list pages.
5. Run a script that pushes the JSON data from the product listing view model into the data layer.

    - You can either add a script tag directly into the view code or link an external JavaScript file.
    - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

```
document.addEventListener("DOMContentLoaded", function (event) {
    dataLayer.push({
        "ecommerce": {
            "impressions": @Html.Raw(Model.GtmProductsJsonString)
        }
    });
});
```

Product clicks

To log clicks of product links:

1. Edit the view model class that you use to display product links (for example on product listing pages):
    a. Add a *string* property, which will store the JSON data of the given product.
    b. Create a JSON string containing the product's data by calling the **GtmProductHelper.MapSKU** and **GtmDataHelper.SerializeToJson** methods, and assign the string to the new property.

```
using CMS.Ecommerce;
using Kentico.Ecommerce;

public class ProductListItemViewModel
{
    public readonly GtmData GtmSKUJson;
    public readonly string GtmSKUJsonString;
    ...

    // Constructor of the view model class representing items in product
lists
        public ProductListItemViewModel(SKUTreeNode productPage,
ProductPrice priceDetail, string publicStatusName)
    {
        // Creates the Google Tag Manager data for the given product
        GtmSKUJson = GtmProductHelper.MapSKU(productPage.SKU);
        GtmSKUJsonString = GtmDataHelper.SerializeToJson(GtmSKUJson);
        ...
    }
}
```

2. Define a JavaScript function for the pages containing product links (for example by adding a script tag to the corresponding views or by linking an external JavaScript file).

    - Within the function, push the data of the clicked product into the data layer. The function must have parameters containing the product JSON data and the URL of the clicked link.
    - Use the *eventCallback* datalayer variable to perform redirection after the product data is sent to Google Analytics.

Function example

```
function productClick(productJson, link) {
    dataLayer.push({
        "event": "productClick",
```

```
        "ecommerce": {
            "click": {
                "actionField": {"list" : "Search Results"},
                "products": [productJson]
            }
        },
        "eventCallback": function () {
            document.location.href = link;
        }
    });
}
```

3. Add an **onclick** script to your product links and call the defined product click function.
    - Get the JSON data of the clicked product from the view model.
    - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

View example

```
@model ProductListViewModel

@foreach (ProductListItemViewModel product in Model.Products)
{
    @* Creates a hyperlink to the product controller to display the product
detail page *@
    string productDetailLink = Url.RouteUrl("Product", new { id = product.
ProductPageID, productAlias = product.ProductPageAlias });
    <a href="@productDetailLink" onclick='productClick(@Html.Raw(product.
GtmSKUJsonString), "@productDetailLink"); return false;'>
    ...
    </a>
}
```

Views of product details

To log data when visitors view your website's product detail pages:

1. Edit the view model class that you use to display product details.
2. Add a new *string* property to the view model.
3. Create a JSON string containing the product's data by calling the **GtmProductHelper.MapSKU** and **GtmDataHelper.SerializeToJson** methods, and assign the string to the new property.

```
using CMS.Ecommerce;
using Kentico.Ecommerce;

public class ProductViewModel
{
    public readonly string GtmSKUJsonString;
        ...

    // Constructor of the view model class used to display product details
        public ProductViewModel(SKUTreeNode productPage, ProductPrice priceDetail)
    {
        // Creates the Google Tag Manager JSON data for the given product and
assigns it to a property
        SKUInfo sku = productPage.SKU;
        GtmSKUJsonString = GtmDataHelper.SerializeToJson(GtmProductHelper.MapSKU
```

```
(sku));
        ...
    }
}
```

4. Edit the view representing your product detail pages.
5. Run a script that pushes the product JSON data from the view model into the data layer.

    - You can either add a script tag directly into the view code or link an external JavaScript file.
    - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

    ```
    document.addEventListener("DOMContentLoaded", function (event) {
        dataLayer.push({
            "ecommerce": {
                "detail": {
                    "actionField": { "list": "Gallery" },
                    "products": [@Html.Raw(Model.GtmSKUJsonString)]
                }
            }
        });
    });
    ```

Additions and removals from a shopping cart

To log data when customers add or remove shopping cart items:

> For more information about shopping cart management in general, see Using a shopping cart on MVC sites.

1. Edit the view model class that you use to display shopping cart content and add a *string* property. The property will store JSON data for added or removed products.

    ```
    public class ShoppingCartViewModel
    {
        public ShoppingCart Cart { get; set; }
        public IEnumerable<ShoppingCartItem> Items => Cart.Items;
        public decimal RemainingAmountForFreeShipping { get; set; }

        // Property containing Google Tag Manager JSON data for added or removed
    products
        public string GtmAddOrRemoveCartItemJsonString { get; set; }
    }
    ```

2. Adjust the controller actions that you use to add or remove products to/from the shopping cart:

    - Create an object with an Anonymous type, and define the data fields that you wish to push into the Google Analytics data layer (see the Enhanced Ecommerce (UA) Developer Guide for details).
    - Get the data of the added or removed product by calling the **GtmProductHelper.MapSKU** method. Use the method's optional *additionalData* parameter to add the quantity of the added/removed product units into the data.
    - Serialize the anonymous type object into a JSON string (for example using the Json.Encode method) and assign the string into the corresponding property of the shopping cart view model. For example, you can use the TempData dictionary to pass the JSON string into another controller action that prepares the shopping cart view model.

    Example - Action for adding items

```
[HttpPost]
public ActionResult AddItem(int itemSkuId, int itemUnits)
{
    // Gets the current user's shopping cart
    ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

    // Adds a specified number of units of a specified product
    cart.AddItem(itemSkuId, itemUnits);

    // Evaluates the shopping cart
    cart.Evaluate();

    // Creates an anonymous type object representing the Google Tag Manager
JSON data for the product addition event
    var addToCartJson = new
    {
        @event = "addToCart",
        ecommerce = new
        {
            currencyCode = cart.Currency.CurrencyCode,
            add = new
            {
                products = new[]
                {
                    GtmProductHelper.MapSKU(SKUInfoProvider.GetSKUInfo
(itemSkuId), new { quantity = itemUnits })
                }
            }
        }
    };

    // Serializes the Google Tag Manager JSON data of the product addition
event into a string
    // Stores the string in the TempData dictionary, which is later used to
prepare the shopping cart view model
    TempData["gtmAddOrRemoveCartItemJsonString"] = System.Web.Helpers.Json.
Encode(addToCartJson);

    // Displays the shopping cart
    return RedirectToAction("ShoppingCart");
}
```

Example - Action for removing items

```
[HttpPost]
public ActionResult RemoveItem(int itemID)
{
    // Gets the current user's shopping cart
    ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

    // Gets the ShoppingCartItemInfo object representing the removed product
    // Note: Must be performed before calling the ShoppingCart.RemoveItem
method
    ShoppingCartItemInfo cartItem = ShoppingCartItemInfoProvider.
GetShoppingCartItemInfo(itemID);

    // Creates an anonymous type object representing the Google Tag Manager
JSON data for the product removal event
```

```
    var removeFromCartJson = new
    {
        @event = "removeFromCart",
        ecommerce = new
        {
            remove = new
            {
                products = new[]
                {
                    GtmProductHelper.MapSKU(cartItem.SKU, new { quantity =
cartItem.CartItemUnits })
                }
            }
        }
    };

    // Removes a specified product from the shopping cart
    cart.RemoveItem(itemID);

    // Evaluates the shopping cart
    cart.Evaluate();

    // Serializes the Google Tag Manager JSON data of the product removal
event into a string
    // Stores the string in the TempData dictionary, which is later used to
prepare the shopping cart view model
    TempData["gtmAddOrRemoveCartItemJsonString"] = System.Web.Helpers.Json.
Encode(removeFromCartJson);

    // Displays the shopping cart
    return RedirectToAction("ShoppingCart");
}
```

Example - Updated action displaying the shopping cart

```
public ActionResult ShoppingCart()
{
    // Gets the current user's shopping cart
    ShoppingCart currentCart = shoppingService.GetCurrentShoppingCart();

    // Initializes the shopping cart model
    ShoppingCartViewModel model = new ShoppingCartViewModel
    {
        // Assigns the current shopping cart to the model
        Cart = currentCart,
        RemainingAmountForFreeShipping = pricingService.
CalculateRemainingAmountForFreeShipping(currentCart),

        // Gets Google Tag Manager JSON data for product addition or
removal events from the TempData dictionary
        GtmAddOrRemoveCartItemJsonString = TempData
["gtmAddOrRemoveCartItemJsonString"] as string
    };

    // Displays the shopping cart
    return View(model);
}
```

3. Edit the view that you use to display shopping cart content.
4. Run a script that pushes the JSON string from the shopping cart view model into the data layer:

    - You can either add a script tag directly into the view code or link an external JavaScript file.
    - Wrap the script code into a condition that checks the JSON string for null or empty values. The condition ensures that the script only runs when displaying the shopping cart after a customer adds or removes products.
    - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

    View example

```
@model ShoppingCartViewModel

...

@if (!String.IsNullOrEmpty(Model.GtmAddOrRemoveCartItemJsonString))
{
    <script type="text/javascript">
        document.addEventListener("DOMContentLoaded", function (event) {
            dataLayer.push(@Html.Raw(Model.
GtmAddOrRemoveCartItemJsonString));
        });
    </script>
}
```

Checkout steps

To log progress of customers through the steps of your website's checkout process:

1. Identify the first page in your checkout process where modifications of the shopping cart content are no longer allowed.
2. Edit the view model class that you use for the given checkout page and add a *string* property. The property will store JSON data for the products in the shopping cart.

    Example

```
public class DeliveryDetailsViewModel
{
    public CustomerModel Customer { get; set; }
    public BillingAddressModel BillingAddress { get; set; }
    public ShippingOptionModel ShippingOption { get; set; }

    // Property containing Google Tag Manager JSON data for products in the
shopping cart
    public string GtmCartItemsJsonString { get; set; }
}
```

3. Adjust the controller actions that you use to display the given checkout page:

    - Get the JSON data for the products in the shopping cart by calling the **GtmProductHelper.MapShoppingCartItems** method.
    - Serialize the shopping cart product data into a JSON string by calling the **GtmDataHelper.SerializeToJson** method, and assign the string into the corresponding property of the view model.

    Example - Action for displaying the delivery details page

```
public ActionResult DeliveryDetails()
{
```

```
    // Gets the current user's shopping cart
    ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

        ...

    // Creates the Google Tag Manager JSON data for the products in the
shopping cart
    IEnumerable<ShoppingCartItemInfo> cartItemInfos = cart.Items.Select
(item => ShoppingCartItemInfoProvider.GetShoppingCartItemInfo(item.ID));
    string gtmCartItemsJsonString = GtmDataHelper.SerializeToJson
(GtmProductHelper.MapShoppingCartItems(cartItemInfos));

    DeliveryDetailsViewModel model = new DeliveryDetailsViewModel
    {
        Customer = new CustomerModel(cart.Customer),
        BillingAddress = new BillingAddressModel(cart.BillingAddress,
countries, null),
        ShippingOption = new ShippingOptionModel(cart.ShippingOption,
shippingOptions),
        GtmCartItemsJsonString = gtmCartItemsJsonString
    };

    // Displays the customer details step
    return View(model);
}
```

4. Edit the view representing the appropriate checkout page:
   - Run a script that pushes the required checkout data into the data layer. Get the JSON data of the shopping cart's products from the view model.
   - You can either add a script tag directly into the view code or link an external JavaScript file.
   - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

```
document.addEventListener("DOMContentLoaded", function(event) {
    dataLayer.push({
        "event": "checkout",
        "ecommerce": {
            "checkout": {
                "actionField": {"step": 1},
                "products": @Html.Raw(Model.GtmCartItemsJsonString)
            }
        }
    });
});
```

5. Edit the views representing the pages used for the remaining steps in the checkout process.
   - For each step, run a script that pushes the required checkout data into the data layer, but without including the *products* field. For example:

```
document.addEventListener("DOMContentLoaded", function (event) {
    dataLayer.push({
        "event": "checkout",
        "ecommerce": {
            "checkout": {
                "actionField": {
                    "step": 2,
                    "option": "@Model.Cart.ShippingOption.
ShippingOptionDisplayName"
```

```
                }
            }
        }
    });
});
```

Purchases

To log purchases (i.e. completed orders) on your website:

1. Identify the page that you display to customers after they successfully complete a purchase on your website (for example a "thank you" page).
2. Set up a view model class for this "thank you" page:
    - Add a *string* property, which will store the JSON data for the purchase.
    - Create a JSON string containing the purchase data by calling the **GtmOrderHelper.MapPurchase** and **GtmDataHelper.SerializeToJson** methods, and assign the string to the property.

    > To get the required *OrderInfo* parameter, call the **OrderInfoProvider.GetOrderInfo** method (from the *CMS.Ecommerce* namespace). The internal *OrderInfo* object has the same ID as the matching *Kentico.Ecommerce.Order* object (this class is used to manage orders in the E-commerce integration API for MVC).

```
using CMS.Ecommerce;
using Kentico.Ecommerce;

public class ThankYouViewModel
{
    public readonly string GtmPurchaseJsonString;

    // Constructor of the view model class for the "thank you" page of the
checkout process
    public ThankYouViewModel(Order order)
    {
        // Gets the internal OrderInfo object for the completed order
        OrderInfo orderInfo = OrderInfoProvider.GetOrderInfo(order.OrderID);

        // Creates the Google Tag Manager JSON data for the purchased order
        GtmPurchaseJsonString = GtmDataHelper.SerializeToJson
(GtmOrderHelper.MapPurchase(orderInfo));
    }
}
```

3. Find an appropriate controller action to display the "thank you" view (for example an action that handles payments). Create an instance of the view model class for the related *Kentico.Ecommerce.Order* object. For more information about order processing, see Creating the preview step in MVC checkout processes.
4. Edit the view that you use to display the "thank you" page.
5. Run a script that pushes the purchase JSON data from the view model into the data layer:

    - You can either add a script tag directly into the view code or link an external JavaScript file.
    - Process the JSON string to ensure that the output is not HTML encoded (for example by calling the HtmlHelper.Raw method).

```
document.addEventListener("DOMContentLoaded", function(event) {
    dataLayer.push({
        "ecommerce": {
            "purchase": @Html.Raw(Model.GtmPurchaseJsonString)
        }
```

```
    });
});
```