

Intilery API

There are a number of ways that data can be sent from a customer's site/app to the Intilery platform.

1. Use the site configuration tool within the Intilery application, you can point and click on your site to define events and data to be collected.
2. Use the javascript API
3. Use the JSON API

Intilery Javascript Tag

Copy the Javascript tag by navigating to Settings --> Install Intilery

Add the Javascript Tag to your site's template page. If you use Google Analytics, place the Javascript Tag above the Google Analytics Tag.

Your site's data collection and events can then be configured using the site configuration point and click tool within the Intilery application.

Intilery Javascript API

In order to use the Javascript API please add the Intilery Javascript Tag using the instructions above.

For each page view that the views on your site, the usage data will automatically be sent to the Intilery system using a HTTP GET request. If you wish to send further data about the user and/or the user's behaviour and cannot do this using the Intilery's Site Configuration then it is possible to send data using the javascript API. Possible scenarios for this are where you use custom AJAX technologies to render your page or other reasons.

How the Asynchronous Syntax Works

The `_itq` object is what makes the asynchronous syntax possible. It acts as a queue, which is a first-in, first-out data structure that collects API calls until `it.js` (Intilery's javascript object) is ready to execute them. To add something to the queue, use the `_itq.push` method.

To push an API call onto the queue, you must convert it from the traditional JavaScript syntax into a command array. Command arrays are simply JavaScript arrays that conform to a certain format. The first element in a command array is the name of the tracker object method you want to call. It must be a string. The rest of the elements are the arguments you want to pass to the tracker object method. These can be any JavaScript value according to the method you are calling asynchronously.

An example of an asynchronous call to `it.js` using `_itq`

```
_itq.push(["_trackPageview"]);
```

One Push, Multiple Commands

Instead of typing `_itq.push(...)` for each call, you can push all of your commands at once. The following code demonstrates this technique.

```
_itq.push([
  ["_trackPageview"],
  ["_trackUserEvent", "sign in", [{"name": "Customer.Email", "value": "me@example.com"}], "Social Login"]
]);
```

Splitting the Snippet

If you prefer to put the Intilery snippet at the bottom of the page (before any the `ga.js` snippet if you use it), you should know that you don't have to put the whole snippet at the bottom. You can still keep most of the benefits of asynchronous loading by splitting the snippet in half—keep the first half at the top of the page and move the rest to the bottom. Because the first part of the tracking snippet has little to no effect on page rendering, you can leave that part at the top and put the part of the snippet that inserts `it.js` at the bottom.

A page with the asynchronous snippet split in half might look like this:

```
<html>

<head>
  <script type="text/javascript">
    var _itq = _itq || [];
    _itq.push(
      ["_trackPageview"],
      ["_trackUserEvent", "sign in", [{ "name": "Customer.Email", "value": "me@example.com" }], "Social Login"]
    );
  </script>
</head>

<body>
  <p>Page Content</p>

  <script src="some_random_script.js"></script>

  <p>Page Content</p>

  <script type="text/javascript">
    _gaq = [] || _gaq; window.INTILERY = {}; INTILERY.gaq = [];
    (function($, _push) { _gaqPush = _push; $.push = function() { INTILERY.gaq.push(arguments[0]); return _gaqPush.apply(_gaq, arguments); }
    _itq.push(["_setAccount", "xxxxx"]);
    (function(){
      var it = document.createElement('script'); it.type = 'text/javascript'; it.async = true;
      it.src = ('https:' == document.location.protocol ? 'https://' : 'http://') + 'www.intilery-analytics.com/rest/it.js';
      var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(it, s);
    })();
  </script>

</body>
</html>
```

Javascript Asynchronous Method Reference

_trackPageview()

Main method for Intilery. Sends the data about the current user and the URL passed to the method to the Intilery system. (note that this method is called automatically by it.js for the current page, use this if you want to log additional page views without the user actually visiting the URL).

```
_itq.push(["_trackPageview", "/home/landingPage"]);
```

_trackUserEvent()

This method is used to send data about the users behaviour on-site to the Intilery system. The event data is mapped to the entities and properties you setup in the Intilery application. Entity and property names are case sensitive, spaces in both are allowed. You can view the entities and entity properties in "Models" section of the Intilery application.

```
_itq.push(["_trackUserEvent", eventAction, [keyValueArray], eventName]);
```

- **eventAction (required)**

The name of the behavioural event, e.g. 'sign-in', 'register', 'add to basket' Note: the eventAction must map to an event that you have configured in the Intilery Application for it to be saved

- **keyValueArray (required)**

An array of 0 or more objects in the form of {"name": "entity.property", "value": "a value"} Note: the name should be in the form of entity.property where the entity and property has been configured in the Intilery application

- **eventName (required)**

The name of the event to track, note this is different from the eventAction. E.g. you may have more than one way a user can login; a usual login form and a social login, both eventActions would be 'sign-in' whilst the eventNames would be 'main' and 'social' respectively.

Example of a `_trackUserEvent` call

```
_itq.push(["_trackUserEvent", "sign in", [{ "name": "Customer.Email", "value": "me@example.com" }], "Social Login"]);
```

Google Analytics Calls

Intilery will automatically clone any calls you make to Google Analytics and send them to Intilery so that you do not have to duplicate the code. In most cases this means that any tracking you have setup for events and ecommerce will automatically be sent to Intilery for you without you having to write any additional code.

The methods that Intilery duplicates are: -

- `_addTrans`
- `_addItem`
- `_trackEvent`
- `_setCustomVar`

Note: This does not affect Google Analytics in any way, it.js clones the data before it reaches Google Analytics.

JSON API

Intilery provides an API for sending customer initiated events, data and page views. Each endpoint is accessed over HTTPS using HTTP BASIC AUTHENTICATION (username/password). You can send a JSON document to the API using curl as in this example:

```
curl -X POST --header "Content-Type:application/json" --user "username-api@intilery.com:password" \
https://www.intilery-analytics.com/api/event -d@register.json
```

Where `register.json` is a file with the following format:

```
{
  "Visit": {
    "Email": "tom@example.com"
  },
  "EventAction": "Register Account",
  "EventName" : "Main Register",
  "EventData": {
    "Customer.First Name": "Tom",
    "Customer.Subscribed": "True"
  }
}
```

note that the EventAction indicates the event that happened, where as the EventName indicates which event triggered it, this allows you to have different type of events that reference the same domain event. E.g. you could have 2 places where a user can register, both are a Register Event but would have different a EventName

The API is set up to accept a flexible JSON structure representing the event data when passed, so the following is also valid:

```
{
  "Visit": {
    "Email": "tom@example.com"
  },
  "EventAction": "Register Account",
  "EventName" : "Main Register",
  "EventData": {
    "Customer":{
      "First Name": "Tom",
      "Subscribed": "True"
    }
  }
}
```

API Endpoints

View

The view endpoint is to indicate that a visitor has viewed a particular page on your site. It only accepts a **single object**.

`https://www.intilery-analytics.com/api/view`

A call to view must pass **VISIT** and **CONTEXT** objects.

A call to view may contain *VIEW*, *CLIENTIP*, *LOCATION* and *USERAGENT* objects.

In most cases you will make the call from your application running on your user's client, e.g. from your phone app running on their phone. In those cases it is not required to pass the *CLIENTIP*, *LOCATION* objects as these will be sent (the IP address) as part of the JSON call.

```
{
  "Visit": {
    "Email": "tom@example.com"
  },
  "Context":{
    "Host":"www.example.com",
    "Path":"/"
  },
  "ClientIP":"192.168.10.100",
  "UserAgent":"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0",
  "Location":{"Latitude":52.3,"Longitude":-2.6}
}
```

Event

The event endpoint accepts information about a user initiated action and triggers an update on the **Customer** based on that information. It accepts either a **single object** or an **array of objects**.

`https://www.intilery-analytics.com/api/event`

A call to event must pass **VISIT**, **EVENTACTION** and **EVENTDATA** and **EVENTNAME** objects.

A call to event may contain *CONTEXT*, *VIEW*, *CLIENTIP*, *LOCATION* and *USERAGENT* objects.

```
{
  "Visit": {
    "Email": "tom@example.com"
  },
  "EventAction": "Sign In",
  "EventName" : "Sign In Homepage",
  "EventData": {
    "Username":"Tom \\\"The Example\\\" Customer"
  },
  "Context":{
    "Host":"www.example.com",
    "Path":"/"
  },
  "ClientIP":"192.168.10.100",
  "UserAgent":"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0",
  "Location":{"Latitude":52.3,"Longitude":-2.6}
}
```

Transaction Events

A **Transaction** can be recorded against a user like an event, but to the endpoint

`https://www.intilery-analytics.com/api/transaction`

For each **item** in the transaction send the items to

`https://www.intilery-analytics.com/api/item`

i.e. POST this to `transaction`

```
{
  "Visit": {
    "Email": "tom@example.com"
  },
  "EventAction": "Check Out",
```

```

"EventName" : "Check Out",
"EventData": {
  "Transaction": {
    "OrderID": "12345",
    "AffiliationID": null,
    "Total": 100.00,
    "Tax": 20.00,
    "Shipping": 0.00,
    "City": "Chester",
    "Region": "Cheshire",
    "Country": "UK",
    "Currency": "GBP"
  }
},
"Context": {
  "Host": "www.example.com",
  "Path": "/"
},
"ClientIP": "192.168.10.100",
"UserAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0",
"Location": {"Latitude": 52.3, "Longitude": -2.6}
}

```

and this to `item` for each item in the transaction

```

{
  "Visit": {
    "Email": "tom@example.com"
  },
  "EventAction": "Purchased Item",
  "EventName" : "Purchased Item",
  "EventData": {
    "Transaction": {
      "OrderID": "12345",
    },
    "Product": {
      "Name": "Han Solo Minifigure"
    },
    "Purchased Item": {
      "SKU": "ABC1234",
      "Price": 25.00,
      "Quantity": 4,
      "Category": "Mammal"
    }
  },
  "Context": {
    "Host": "www.example.com",
    "Path": "/"
  },
  "ClientIP": "192.168.10.100",
  "UserAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0",
  "Location": {"Latitude": 52.3, "Longitude": -2.6}
}

```

Data

The data endpoint updates models with no update to the **Customer**. It accepts either a **single object** or an **array of objects**.

<https://www.intilery-analytics.com/api/data>

A call to data should only contain data representing the entity to be updated, this data is interpreted in the same way as **EVENTDATA** but without being nested.

```

[
  {
    "Product": {
      "Name": "Han Solo Minifigure",
      "Brand": "Lego",
      "Price": 7.99
    }
  },
  {
    "Product.Name": "Luke Skywalker Minifigure",
    "Product.Brand": "Lego",
    "Product.Price": 6.99
  }
]

```

```

    },
    {
      "Brand": {
        "Name": "Lego",
        "Website": "http://www.lego.com/"
      }
    }
  ]
}

```

JSON Objects

Each call should pass a **JSON** formatted object or array. Although each object is shown in camel case in the examples below the API is not case sensitive.

Where a **timestamp** is used as a data type it is the number of seconds since 1970-01-01 T00:00:00Z known as the **Unix Time**. The timestamp can be sent in seconds or milliseconds. The Intilery platform is intelligent enough to tell the difference.

Visit

A Visit object is used to identify an event or a view with a particular visitor or **Customer**. Where the **Customer** is known then the Visit object can contain the **Customer** email. If the email isn't known then the Visit is identified by a VisitorID, InitialVisit and CurrentSession.

```
"Visit":{"Email":"tom.mcmillen@intilery.com"}
```

```
"Visit":{"VisitorID":123456789,"InitialVisit":1377990000,"CurrentSession":1377990000}
```

- **VisitorID** is a number that should be generated by you to identify that visitor.
- **InitialVisit** is a timestamp for the first time a visitor was identified.
- **CurrentSession** is a timestamp for the start of the current session.

The **VisitorID** and **InitialVisit** uniquely identify a visitor.

The **CurrentSession** groups all the visitors actions into a single session.

If a visit is identified by the **Email**, then the Intilery platform will assign a **VisitorID**, **InitialVisit** and **CurrentSession**, with a new session being generated after every 30 minutes of inactivity by that user.

Context

A Context object provides information on where a event or view happened.

A Context object must have a **Host** and **Path**,

A Context object may have a *ReferrerHost* and a *ReferrerPath*.

```
"Context":{"Host":"www.intilery.com","Path":"/"}
```

```

"Context":{
  "Host":"www.intilery.com",
  "Path":"/",
  "ReferrerHost":"www.google.com",
  "ReferrerPath":"/?q=customer+engagement"
}

```

View

Provides additional information about the client when they viewed a page or carried out an action.

A View object may have any of the following fields *Title*, *PageDepth*, *Encoding*, *Language*, *JavaEnabled*, *ScreenX*, *ScreenY*, *ViewX*, *ViewY*, *QueryString*.

```

"VIEW":{
  "Title":"Home",
  "PageDepth":3,
  "Encoding":"UTF-8",
  "Language":"en-GB"
}

```

```
}

```

- **Title** should be the title of the page being viewed.
- **PageDepth** is the number of the page view in the session.
- **Encoding** is the text encoding used to display the page.
- **Language** is the language and locale represented by the [ISO 639-1](#) and [ISO 3166-1 alpha-2](#) codes.
- **JavaEnabled** is either true or false (default) to indicate if Java is enabled.
- **Screen[XY]** is the size of the screen.
- **View[XY]** is the size of the window/view the visitor has.
- **QueryString** is any additional data on the end of the **Contexts** path after a `?`.

ClientIP

The IPV4 IP address of the client request. It is represented as a string.

```
"ClientIP": "192.168.8.100"

```

If this is passed and no **LOCATION** object is sent, the Intilery platform will try to infer the users location from the IP address passed.

Location

The Location object can be passed if you know the location the request comes from.

A Location object must have a **Latitude** and a **Longitude**.

A Location object may have a *Country*, *Region* and *City*.

```
"Location": {"Latitude": 52.3, "Longitude": -2.6}

```

```
"Location": {"Latitude": 52.3, "Longitude": -2.6, "City": "Chester", "Region": "Cheshire", "Country": "GB"}

```

- **Latitude** should be a number between -90.0 and +90.0.
- **Longitude** should be a number between -180.0 and +180.0.

If you only pass **Latitude** and **Longitude** the Intilery platform will look up the closest Country, Region and City data.

UserAgent

The User Agent of the visitors device. It is represented as a string.

```
"UserAgent": "Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_4 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B

```

EventAction

For an event, this identifies the type of event action the visitor is initiating. This should correspond to one of the events you have defined for your integration. It is a string.

EventName

For an event, this differentiates the different events that can happen for the same actual event, e.g. Sign-In on the homepage or Sign-In a different way - both are a Sign-In event with a different EventName

```
"EventAction": "register account"

```

EventData

For an event, or as an unqualified object when passed to the data endpoint, the EventData represents the updates to the models to be carried out. When part of an event, then **Customer** segments will be updated.

Pass each entity to be updated as a key in the format `"Entity.Field"`, or the API can interpret the data as a set of nested objects, so both of the examples below are valid.

```
{
  "EventData":{
    "AddToBasket.SKU":"1234",
    "AddToBasket.Quantity":2,
    "AddToBasket.Price":7.99,
    "Product.Name":"Han Solo Minifigure",
    "Product.Brand":"Lego"
  }
}
```

```
{
  "EventData":{
    "AddToBasket":{
      "SKU":"1234",
      "Quantity":2,
      "Price":7.99
    },
    "Product":{
      "Name":"Han Solo Minifigure",
      "Brand":"Lego"
    }
  }
}
```