Conversion Path

Setup Guide & Documentation

What is Conversion Path?
- It's Easy to Get Started
- It's Easy to Maintain
- It Captures All Marketing Touchpoints
- It Restores Functionality Removed by Marketo

Overview
What’s new in this version?
A Simple Example
How It Works
- Default Field Mapping
- Default Unpaid Referrer Classification
- Unknown classification
- Limitations

Implementing Conversion Path
- Adding Conversion Path JavaScript to Your Web Pages
- Custom Field Mapping

Integrate with a non-Marketo Form
- Overview
- Enabling Custom Events
- Firing RC_TouchHistoryFormReady
- Firing RC_TouchHistoryRecordable
- Listening for RC_TouchHistoryRecorded
- Preventing Hidden Field Creation

Advanced Script Configuration
    - Custom Query String and Hostname Parsers
        - when (required)
        - matches: optional, default "(.+)"
        - set: required
        - to: optional, default "$1"
- Technical Details
- How to Verify that the JavaScript is Successfully Implemented
- Tracking Activity Across Subdomains

What is Conversion Path?

For digital marketers, effectively and reliably collecting marketing attribution data can be one of the most difficult tasks.

Most companies have not prepared themselves for success in this area as they capture partial or incorrect (and misleading) information. Another common mistake is capturing either first or last touch information and basing attribution solely on these data points.[1] Making decisions based on faulty, incomplete data points can lead to budget misallocation, a mistrust and frustration within multiple departments of your organization.

Conversion Path for Marketo, a solution provided by Romanoff Consulting, makes capturing marketing touchpoint information easier than ever before.

Here is why Conversion Path is awesome:

It's Easy to Get Started

Capturing ***first touch*** data (the marketing activity that first brings a prospect to your website) is certainly possible using Marketo smart lists and triggers but there is a significant build-out time associated with such a project. However, reliably capturing the ***acquisition touch*** (the touch point immediately preceding an initial conversion, or a form fill) or ***recent touch*** (the most recent touchpoint, regardless if a lead is known or unknown) is overly complicated and unreliable using similar methods.

It's Easy to Maintain

By removing the classification from Marketo programs, you eliminate the headache of reviewing dozens of marketing automation programs in an attempt to figure out why a lead was attributed to a certain source. Companies with large databases and high volume web traffic will benefit from a significant reduction in the number of triggered campaigns that would otherwise be required to perform this type of classification, thus improving Marketo performance.

It Captures All Marketing Touchpoints

Stop relying on first touch or last touch information to calculate your marketing ROI! Conversion Path allows you to leverage multiple marketing touchpoints to see what's working at different stages in the buyer's journey.

[1] The problem with exclusive use of either first or last touch information is apparent - you are assigning all of the credit to a marketing campaign or activity that influenced a portion of the attributed marketing success, leading to decisions based on incomplete information.

Conversion Path can be configured to capture custom events that are relevant to your prospects.[2]

It Restores Functionality Removed by Marketo

In February 2016, Marketo rolled out Munchkin version 2 which changed our ability to update attributes of an unknown Lead. This means that programs created by savvy Marketo users to set first or last touchpoint data are, in many cases, broken or seriously impaired.

To learn more about the recent changes to Marketo Munchkin, you can review Marketo's documentation.

Overview

The Conversion Path solution from Romanoff Consulting leverages proprietary JavaScript code, which can easily be added to the header or footer of your website(s).

By default Conversion Path allows you to store **First Touch**, **Acquisition Touch** and **Recent Touch** information but it can be configured to store custom events that are relevant to your prospect's buying journey.

> **First Touch** - this is the first marketing activity that brought a new prospect to your website.
>
> **Acquisition Touch** - this is the marketing activity that a new prospect engaged with immediately preceding a form fill (i.e. becoming a Known prospect in Marketo).
>
> **Recent Touch** - this refers to any marketing activity that the prospect engaged with most recently, even after filling out a form and becoming a Known prospect. This field is constantly updated with the most recent campaign touchpoint.

[2] To learn more about this feature, speak to your primary contact at Romanoff Consulting.

A Simple Example

A new visitor comes to the company's website from a Google Adwords advertising campaign; the URL is:

http://mysite.com/page1?utm_medium=Search&utm_source=Google&utm_campaign=Candy%20Ebook&utm_term=chocolate&utm_content=ebook

The person clicks around the site for a while, but ultimately does **not** fill out a form. Since this is the initial touchpoint that the person had with our site, we want to capture this information in our First Touch fields: ***First Medium***, ***First Source***, ***First Campaign***, ***First Term***, and ***First Content***. In other words, we want…

…to parse the value from...	**...and write the value to the field...**
utm_channel=**Paid**	**First Channel** = Paid
utm_medium=**Search**	**First Medium** = Search
utm_source=**Google**	**First Source** = Google
utm_campaign=**Candy%20Ebook**	**First Campaign** = Candy Ebook
utm_term=**chocolate**	**First Term** = chocolate
utm_content=**ebook**	**First Content** = ebook

A month later, the same visitor comes across a retargeting ad. This person clicks the ad and comes back to our site. The link that they click is:

http://mysite.com/page1?utm_channel=Paid&utm_medium=Retargeting&utm_source=Adroll&utm_campaign=Ice%20Cream%Webinar&utm_term=ice%20cream

This person navigates from the initial landing page to a few different product pages, and eventually fills out a form, becoming a known lead in Marketo. Since the retargeting ad was the last marketing activity that brought the person to our website prior to filling out a form, we want to store that to our fields that store lead acquisition data.

…to parse the value from...	**...and write the value to the field...**
utm_channel=Paid	**Acquisition Channel** = Paid
utm_medium=Retargeting	**Acquisition Medium** = Retargeting
utm_source=Adroll	**Acquisition Source** = Adroll
utm_campaign=Ice%20Cream%20Webinar	**Acquisition Campaign** = Ice Cream Webinar
utm_term=ice%20cream	**Acquisition Term** = ice cream
	Content = ~

A few months later, the same visitor comes back. This time they engage with one of our display ads. We will not want to update any of our First Touch or Last Touch campaigns since those are locked down, however we do want to update our Recent Campaign field to show the last touchpoint that this person had with our website.

This time, their link is:

http://mysite.com/page1?utm_medium=Display&utm_source=Yahoo&utm_campaign=June%20promo&utm_content=vertical-ad

Now we want...

…to parse the value from...	**...and write the value to the field...**
utm_campaign=**June%20promo**	**Recent Source** = June promo

How It Works

Conversion Path uses proprietary JavaScript code to identify and extract Google Analytics URL parameters and write those values to the right fields at the right time. Conversion Path also classifies referrer information when URL parameters are not present, typically associated with unpaid earned and unpaid owned marketing activities (e.g. direct traffic, organic search visits, social media referrals, etc.)

Default Field Mapping

Unless custom fields are defined, Conversion Path will attempt to update a default set of fields, specified below.

Note: The default field mapping can easily be customized to compensate for your existing or preferred field names.

URL Parameter	***Default Marketo Field Name***	***Default SFDC Field Name***
utm_channel	First Channel Acquisition Channel	RC_First_Channel__c RC_Acquisition_Channel__c
utm_source	First Source Acquisition Source	RC_First_Source__c RC_Acquisition_Source__c
utm_medium	First Medium Acquisition Medium	RC_First_Medium__c RC_Acquisition_Medium__c
utm_adgroup	First Adgroup Acquisition Adgroup	RC_First_Adgroup__c RC_Acquisition_Adgroup__c
utm_campaign	First Campaign Acquisition Campaign	RC_First_Campaign__c RC_Acquisition_Campaign__c
utm_content	First Content Acquisition Content	RC_First_Content__c RC_Acquisition_Content__c
utm_term	First Term Acquisition Term	RC_First_Term__c RC_Acquisition_Term__c

Default Unpaid Referrer Classification

When URL parameters are not present Conversion Path attempts to classify traffic to some predefined values. Our default configuration is constantly being improved but there are many domains that cannot be classified by default. When this happens, we write the full URL to the Source field(s).

The following table shows the default mapping of domain referrer information and how we classify the appropriate First or Acquisition fields. Note that these can be overwritten using custom rules which are discussed in greater detail later in this document.

Note that in this context, an asterisk (*) represents a wildcard, meaning any combination of characters. For example Google has hundreds of active domains, so our example (*.google.*) means that we will match any subdomain and top level domain combination for the web company. For example, blog.google.com, gmail.google.in, hi.google.cn and any other combination will be matched correctly.

Web URL(s)	Default Classification

*.*yourdomain*.*
None
 	
Note: Your company's primary domain(s) will be added during the setup and configuration process.
	Channel = UnpaidEarned
	Medium = Direct

.linkedin.
.lnkd.in/
	Channel = UnpaidEarned
	Medium = Social
	Source = LinkedIn

.facebook.
.fb.com/
	Channel = UnpaidEarned
	Medium = Social
	Source = Facebook

.twitter.
.t.co/
	Channel = UnpaidEarned
	Medium = Social
	Source = Twitter

.google.
.g.co/
	Channel = UnpaidEarned
	Medium = Search
	Source = Google

.bing.	Channel = UnpaidEarned
	Medium = Search
	Source = Bing

.yahoo.	Channel = UnpaidEarned
	Medium = Search
	Source = Yahoo

Unknown classification

From time to time there are scenarios where Conversion Path will not be able to classify referral traffic. By default we will write the value of “Unknown” to the First, Acquisition or Recent touch fields. This behavior can be modified (to leave the fields blank, for example) from the configuration settings.

Limitations

As with any web analytics product or service, Conversion Path is constrained to the data that it can collect and the web sessions that it can reliably connect. Conversion Path cannot, for example, reliably associate a person’s desktop browser session with a session from their mobile device. Additionally, anonymous browsing and duplicate records poses a problem that is outside the scope of this solution.

Implementing Conversion Path

Adding Conversion Path JavaScript to Your Web Pages

To add Conversion Path tracking to your web pages, include the following script tag:

```
<script type="text/javascript"
src="//du4pg90j806ok.cloudfront.net/js/touch-history/dist/conversionpath-0.
4.3.min.js"></script>
```

On pages that have a Marketo form, the above `<script>` must appear *after* the <script> that loads Marketo's `forms2.min.js`.

On pages that have a custom form and use Conversion Path's `customEvents` model, the <script> must be placed appropriately to receive custom events: do not send custom events to the page before Conversion Path has loaded and started listening, or it will not hear them! Generally, this means it should be at as high up in the <head> as possible, but your web developer may relocate it based on technical knowledge of your site.

On pages that don't have *any* forms, the Conversion Path <script> can be placed anywhere in the document.

Custom Conversion Path configuration(s)

To customize a Conversion Path installation, add a JSON object (not merely a JS object literal, but valid JSON!) in one of 2 places:

1. *Inside* the the remote `<script>` tag that loads the `conversionpath.js` library from the CDN. This is a somewhat arcane method, admittedly deprecated in the latest HTML5 draft, that we like because it's easier to manage.
2. Inside a local `<script>` with these exact attributes, placed *above* the `<script>` that loads the library.

   ```
   <script type="application/json" id="CP_TouchHistoryOptions"></script>
   ```

   Note the `type` attribute closely as it indicates to the browser this is a non-executable script (an HTML5 "data block").

Here's an example using the 1st method. You can use fields in your Marketo/SFDC instance that differ from the default field names by defining a `nameMappings` property within the config object, like so:

```
<script type="text/javascript"
src="//du4pg90j806ok.cloudfront.net/js/touch-history/dist/conversionpath-0.
4.3.min.js">
{
         "nameMappings": {
              "RC_First_Channel__c": "My_Custom_First_Channel__c",
              "RC_First_Medium__c": "My_Custom_First_Medium__c",
              "RC_First_Source__c": "My_Custom_First_Source__c",
              "RC_First_Campaign__c": "My Custom_First_Campaign__c",
              "RC_First_Term__c": "My_Custom_First_Term__c",
              "RC_First_Content__c": "My_Custom_First_Content__c",
              "RC_Acquisition_Channel__c": "My_Custom_Channel__c",
              "RC_Acquisition_Medium__c": "My_Custom_Medium__c",
              "RC_Acquisition_Source__c": "My_Custom_Source__c",
              "RC_Acquisition_Campaign__c": "My_Custom_Campaign__c",
              "RC_Acquisition_Term__c": "My_Custom_Term__c",
              "RC_Acquisition_Content__c": "My_Custom_Content__c",
              "RC_Recent_Source__c": "My_Custom_Recent_Source__c"
         }
}
</script>
```

Or the same result using the 2nd method:

```
<script type="application/json" id="CP_TouchHistoryOptions">
{
         "nameMappings": {
              "RC_First_Channel__c": "My_Custom_First_Channel__c",
              "RC_First_Medium__c": "My_Custom_First_Medium__c",
              "RC_First_Source__c": "My_Custom_First_Source__c",
              "RC_First_Campaign__c": "My Custom_First_Campaign__c",
              "RC_First_Term__c": "My_Custom_First_Term__c",
              "RC_First_Content__c": "My_Custom_First_Content__c",
              "RC_Acquisition_Channel__c": "My_Custom_Channel__c",
              "RC_Acquisition_Medium__c": "My_Custom_Medium__c",
              "RC_Acquisition_Source__c": "My_Custom_Source__c",
              "RC_Acquisition_Campaign__c": "My_Custom_Campaign__c",
              "RC_Acquisition_Term__c": "My_Custom_Term__c",
              "RC_Acquisition_Content__c": "My_Custom_Content__c",
              "RC_Recent_Source__c": "My_Custom_Recent_Source__c"
         }
}
</script>
<script type="text/javascript"
src="//du4pg90j806ok.cloudfront.net/js/touch-history/dist/conversionpath-0.
4.3.min.js"></script>
```

Multiple configurations on the same page using Configuration Zones

New in v0.4.0, Conversion Path supports *Configuration Zones* (CZs), which allow parts of the configuration to be chosen dynamically depending on factors from the runtime browser environment. For example, different forms on the page (such as forms from different Marketo instances) can have different sets of hidden fields..

To use Configuration Zones, add a special JSON Array property, `@zones`, to a config section. For the (imaginary) section called `balloonOptions`:

```
{
  "balloonOptions" : {
    "liftingGas" : "helium",
    "@zones" : []
  }
}
```

Each item in the `@zones` Array (empty so far) is an object with:

1. the property `@zoneType`, the type of zone you wish to match on (see note below on supported types)
2. the property `@zoneValue`, the value of `@zoneType` that must be matched to select this zone
3. any properties -- with the same naming rules/expectations as for the parent JSON object -- to be applied *to the matched zone only*

Here, the config has had zone-specific `balloonOptions` added:

```
{
  "balloonOptions" : {
    "liftingGas" : "helium",
    "@zones" : [
    {
      "@zoneType" : "munchkinId",
      "@zoneValue" : "aaa-123-ccc",
      "liftingGas" : "hydrogen"
    },
    {
      "@zoneType" : "munchkinId",
      "@zoneValue" : "ddd-456-fff",
      "sniffingGas" : "ammonia"
    }
    ]
  }
}
```

With this config, if you're in the zone where `munchkinId` matches `"aaa-123-ccc"` the effective config section would be:

```
{
  "liftingGas" : "hydrogen"
}
```

While if you're in the zone where `munchkinId` matches `"ddd-456-fff"` the effective section would be:

```
{
  "liftingGas" : "helium",
  "sniffingGas" : "ammonia"
}
```

As you can see, the global options are merged with any zone-level options by default (see **Configuration Zone options** below) with the zone winning in case of a conflict.

And if you didn't have any matching zone, the section would be from the global level only:

```
{
  "liftingGas" : "helium"
}
```

Configuration Zone options

In the root config, the property `zones` controls CZ operation. It currently supports one Boolean property `mergeParentWithZones`, defaulting to `true`. If overridden to `false`, as in

```
{
  "zones" : {
    "mergeParentWithZones" : false
  }
}
```

then global (non-zone-specific) options will *not* be merged with all zone-specific options. In the above example, the `liftingGas` property would not be present in the 2nd zone unless it were *also* added at the zone level.

Current Configuration Zone restrictions

The first release of CZs is restricted to only certain config sections and types:

1. the only `@zoneType` supported is `munchkinId`, which is a case-insensitive match against a Marketo instance's Munchkin ID
2. only the `nameMappings` section is checked for `@zones`

Integrate with a non-Marketo Form

While Conversion Path was originally designed to integrate with Marketo Forms 2.0 forms (both embedded and on Marketo-hosted LPs), we also offer a robust event model to allow integration with any 3rd-party forms library.

Overview

Integration with non-Marketo forms, while not extraordinarily complex, does require an experienced web developer who understands your forms. There's no drop-in code that can automatically and correctly integrate with the unlimited range of forms out there (one-size-fits-all code that attempts to do so always fails, depriving you of mission-critical data). Instead of prefab code, we receive and send standard W3C Custom Events at a couple of key points.

The key points we monitor are:

1. *When the form is available for appending hidden fields.* Since forms may not exist in the page at the time that Conversion Path loads -- very common with 3rd-party forms libraries that load form metadata via Ajax and render the <form> into the page asynchronously -- we need to be notified that there's a form (or forms) to which you want to attach tracking fields. This event is `RC_TouchHistoryFormReady`. (With Marketo forms, we listen for their built-in `MktoForms2.whenReady` event, which serves the same purpose.)
2. *When the form has been successfully submitted to your server*, so we can commit the latest saved values to our history. This is critically different from a primitive `submit` event, since with 3rd-party data validation and enrichment plugins -- not to mention the occasional server error that may cause a form post to fail -- the mere presence of an initial `submit` event in no way means that the current values of the form have actually been accepted and saved. The event we listen for here is `RC_TouchHistoryRecordable`. (The Marketo equivalent is `form.onSuccess`, which is only fired when Marketo's servers have successfully saved form data.)
3. *When history tracking is all done and normal form/page actions can continue.* This is an event *we* fire and *you* listen on: `RC_TouchHistoryRecorded`.

Enabling Custom Events

To enable custom events, override the `events.model` property in the JSON config object:

```
<script type="text/javascript"
src="//du4pg90j806ok.cloudfront.net/js/touch-history/dist/conversionp
ath-0.4.3.min.js">
{
        "events": {
           "model" : "customEvents"
        }

}
</script>
```

The default value of `events.model` is `"mktoForms"`.

Firing RC_TouchHistoryFormReady

Here's a very simple example of a generic HTML form and a companion call you'd make to fire the `RC_TouchHistoryFormReady` event:

```
<form id="genericForm">
```

```
    <input name="input1" type="text">
    <input type="submit">
</form>
<script>
var ce = new CustomEvent("RC_TouchHistoryFormReady", {
    bubbles: true
});
document.getElementById('genericForm').dispatchEvent(ce);
</script>
```

As you can see, here we've simply waited for the form to be in the DOM and fired the event. In the real world, you will likely wait for other events, such as an Ajax fetch of form data and/or metadata, before sending us the Conversion Path event.

The `bubbles: true` is needed because Conversion Path listens on the top-level window so it can hear events from anywhere in the document. You can dispatch the event on `window` if you prefer, so you don't need `bubbles`, but the flipside is you have to include a reference to the form (CSS selector) so we can find the form in question:

```
<form id="genericForm">
    <input name="input1" type="text">
    <input type="submit">
</form>
<script>
var ce = new CustomEvent("RC_TouchHistoryFormReady", {
    detail: {
        formSelector: '#genericForm'
    }
});
window.dispatchEvent(ce);
</script>
```

An advantage of firing on `window` is that if you have multiple forms on the page, you can include them all in `detail.formSelector` and Conversion Path will attach to all of them. It's up to you! And you can also use whatever wrapper you want around Custom Events, such as one that comes with your favorite JS framework. As long as it ends up emitting a true Custom Event, it'll be fine.

Firing RC_TouchHistoryRecordable

Firing `RC_TouchHistoryFormReady` tells Conversion Path it has the go-ahead to add fields to the form(s) that fired the event. The hidden fields are added immediately (when we receive the event) so it's a one-and-done process.

Somewhat more complex is making sure you signal to TH that it can *commit* (save permanently) its current history values to a browser cookie. This can only be done if your forms library is as *sure as possible* that the current form has been -- or will be -- saved to your server.

Hopefully, your library can fire the event the moment *after* values are successfully posted to your server (like Marketo's Forms 2.0 library does when its Ajax POST gets HTTP 200 OK) but sometimes you can only know that you are *about* to send the values, because the user is going to navigate away from the page on form post. In the case of a barebones HTML form, for example, it's impossible to send an update to an in-memory JavaScript object after the form posts, because at that point the JS object will have been destroyed along with the rest of the page.

So, if possible, you should fire `RC_TouchHistoryRecordable` *directly after* a successful form submission is confirmed, but if that isn't possible, fire it *directly before.* In the simple (and quite fragile) example below, we are using the suboptimal method of firing before:

```
document.getElementById('genericForm').addEventListener('submit',
function(e) {
      window.dispatchEvent(new
CustomEvent("RC_TouchHistoryRecordable"));
      e.stopPropagation();
      e.preventDefault();
});

document.getElementById('genericForm').addEventListener('RC_TouchHist
oryRecorded', function(e) {
      console.log('ready to submit form', this.id, 'without further
interruption');
      this.submit();
});
```

Here, we hook the standard `submit` event, but we promptly stop event propagation (plus the default form action) and dispatch `RC_TouchHistoryRecordable` instead. This signals to Conversion Path that it should save its current values. TH then "calls back" to your form with the `RC_TouchHistoryRecorded` event, which tells you that you're good to finish submitting the form.

What's extremely relevant is that `stopPropagation()` does not stop other `submit` listeners -- those bound at the <form> element level -- from hearing the `submit` event and doing whatever else they want. A later listener might happily submit the form even though you don't want it to. Or vice versa: a later listener may redirect the page even though you wanted to send the form when all listeners finished. The fact is you can't ensure cooperation among submit listeners, unless you happen to author all the code. So, while this contrived example may function acceptably, it should be easy to see that entrusting history tracking to a chain of disparate event listeners is very risky. That's why we strongly encourage only firing `RC_TouchHistoryRecordable` once all validation steps and the on-the-wire submission itself are complete. Here's an example using an imaginary forms library, MyNiceForms:

```
MyNiceForms.on('ajaxComplete', function(e) {
      window.dispatchEvent(new
CustomEvent("RC_TouchHistoryRecordable"));
      return false;
});

document.getElementById('genericForm').addEventListener('RC_TouchHist
oryRecorded', function(e) {
      document.location.href="ThankYouPage.html";
});
```

You should be able to adapt to the methods and events available in any forms library.

Listening for RC_TouchHistoryRecorded

After your forms library fires `RC_TouchHistoryRecordable`, listen for the response event `RC_TouchHistoryRecorded` before proceeding. "Proceeding," of course, means something different depending on your environment:

- If you fired `RC_TouchHistoryRecordable` *after* the form was submitted (the preferred event order), then proceeding means redirecting the page using `location.href,` or perhaps modifying the current page's DOM via Ajax fetch.
- If you fired `RC_TouchHistoryRecordable` *before* the form was submitted, then proceeding means submitting the form.

Preventing Hidden Field Creation

By default, Conversion Path injects a hidden INPUT element into your form for each field if is not already present. If you wish to disable this behavior and only use those <INPUT type="hidden"> elements that already exist, set the config value `events.injectHiddenFields` to `false`.

```
        "events": {
           "model" : "customEvents",
            "injectHiddenFields" : false
        }
```

Note this value is only relevant when using Custom Events. In the default Marketo Forms event model, hidden fields are always added.

Advanced Script Configuration

The following are just a few ways to further extend Conversion Path. Do you have an idea to make Conversion Path even better? Contact joe@rc.solutions

Custom Query String and Hostname Parsers

You can use our robust domain-specific language (DSL) to define rules for mapping specific domains, referrals, or query strings to field values. Conversion Path internally uses the same DSL to store our default settings, which you can see at this Github Gist.

The DSL is used within the `stageMappings` property of the JSON config object, as in the example below.

```
<script type="text/javascript"
src="//du4pg90j806ok.cloudfront.net/js/touch-history/dist/conversionp
ath-0.4.3.min.js">
{
   "stageMappings" : {
      "initial" : [
         {
            "when" : "{referrer.query.my_channel}",
            "matches" : "(.+)",
            "set" : "RC_First_Channel__c",
            "to" : "$1"
         }
      ],
      "rolling" : [
         {
            "when" : "{referrer.query.my_channel}",
            "matches" : "(.+)",
            "set" : "RC_Acquisition_Channel__c",
            "to" : "$1"
         }
      ],
      "recent" : [
         {
            "when" : "{current.query.my_campaign}",
            "matches" : "(.+)",
            "set" : "RC_Recent_Campaign__c",
            "to" : "$1"
         }
      ]
   }
}
</script>
```

As you can see, the DSL emulates a fluent, English-like interface: “When *A* matches *B*, set *C* to *D*.” You can add entries for each touch stage: `initial`, `rolling` (ongoing until acquisition), and `recent` (latest) .

`when:` required

The `when` property of each entry uses industry-standard URI Template syntax (published at https://tools.ietf.org/html/rfc6570) to specify matching targets. You can match against properties of the current URL (the `current` top-level property) or the referrer URL (the `referrer` top-level property) or other document contents (see **Avilable URI Template variables** below).

`matches:` required, typically `"(.+)"`

The `matches` property, as its appearance suggests, is used to create a JS regular expression (RegExp) object. It’s automatically wrapped in start and end anchors (`^` and `$`) so be careful to take that into account in your custom entries. In the examples above, the resulting regex is `/^(.+)$/`, equivalent to “is full match.” To get a “contains match” instead, add wildcards.

`set:` required

The `set` property specifies the form field name. Note that custom `stageMappings` rules are executed *before* custom `nameMappings` rules, so the field in `set` might be superseded by a custom fieldname mapping.

`to:` optional, default `"$1"`

The `to` property specifies the value to write to the form field. As with `matches`, we leverage the JavaScript RegExp engine for replacement. The default `"$1"` means the first/only captured expression. You may find it useful to hard-code a value here. For example, if you look at the default ruleset, when we detect a Twitter referrer, we write the static string `"Twitter"` as the source, rather than using any part of the URL.

Available URI Template Variables

`{current.query}` and `{referrer.query}`

As should be clear from the examples above, these are JS objects with a property for each query param.

`{current.domain}` and `{referrer.domain}`

The respective HTTP hostnames.

`{current.registeredDomain}`

This special property gives you access to the domain you'd purchase from a registrar, i.e. the registered domain, which can be very handy for matching across all subdomains of a parent domain.

`{current.meta}`

The parent object has properties for the name of every `<meta>` tag in the document (the name being the `name`, `property`, or `itemProp` attribute, whichever is first and non-empty).

Namespace-style names with colons like `"og:title"` are *unflattened into nested objects and properties* and the colons are removed.

So

```
    <meta name="standard:link:ref">
```

is referenced as

```
    {current.meta.standard.link.ref}.
```

`{current.head}`

The object has properties for interesting child elements of the document `<head>`. As of v0.4.0 only the text of the HTML `<title>` tag, `{current.head.title}`, is supported.

`{current.cookie}`

All currently accessible cookies.

Valid JSON found in cookie values is always deserialized. So a cookie named `jsoninside` with the raw stored value (string value, like all cookies):

```
{"hello":"there"}
```

means the template variable

```
{current.cookie.jsoninside.hello}
```

will be available.

Substring extraction (advanced)

In addition to simple token replacement/concatenation, URI Templates also support substring extraction (`{current.domain:2}` gives the first two characters of `{current.domain}`) and other goodies, although you can get much of the same functionality by using basic replacement plus a more complex `matches` regex).

Other technical details

The script is served from our CDN (AWS CloudFront) for high performance, though during the testing period it only has a 2-minute expiration to allow us to make changes quickly (in other words, you'll get a fresh copy after 2 minutes, even without changing the version number). For final rollout, we'll be setting the expiration to at least a full day, so you'd simply bump the version number to get the latest file if changes are made.

The script is self-initializing at present: you don't need to have a line like TouchHistory.init() or anything like that. To accommodate on-the-fly changes, though, it might be preferable to have an init() step. That way, your side can change a field in Marketo/SFDC anytime and we don't need to change the code at all (imagine if you want to change utm_campaign to go to New_Campaign_Field_2__c and so on, or adding new UTM params entirely).

While testing in Incognito/Private/InPrivate browsing mode is always far preferable, you can clear Conversion Path by looking for any cookies that start with

`rcTouchHist_` (old legacy mode)
`cpTouchHist_` (default for v0.3.9+)

How to Verify that the JavaScript is Successfully Implemented

When the script is running, it'll log messages to the browser console:

```
Conversion Path: getting START history
Conversion Path: overwriting ROLLING history
Conversion Path: setting RECENT history
Conversion Path: Adding fields to Marketo form 1339
Conversion Path: Adding fields to Marketo form 1442
```

When there's a Marketo form on the page, you should always see a corresponding log line for the form, either "Adding fields to..." or "No fields to add to...".

You'll also see a verbose dump of fields as they are stored and/or added to forms.

Logs may be suppressed using the following root config option:

```
{
  "nolog" : true
}
```

If you require *completely* silent operation, you must preset a global JS variable (otherwise there will be a couple of log entries *before* the options are consulted, i.e. logging the options-loading process itself!).

```
var CP_TouchHistoryOptions_nolog = true
```

Tracking Activity Across Subdomains

Conversion Path automatically sets its cookies at the highest possible level, so the cookies get shared across sites under a parent domain.

Therefore if a prospect enters the parent domain via ***blog****.mysite.com?utm_campaign=AAA*, if I fill out a form on ***www****.mysite.com* it will know that my `Initial Campaign`='AAA'.