API Workflow: Timeline

Table of Contents

Summary

Timelines record various types of temporal events, each with its own different set of data attributes. There is no limit to the size or type of timeline you can define. While a user event timeline is typical, others, such as campaign or point of sale timelines, can also be created. Having multiple timelines allows you to keep events logically separated along service boundaries and not easily intermingled.

The Timeline APIs are organized into three basic sets of functionality:

- Timeline Service API allows you to build and manage an event timeline service. It offers the ability to define different kinds of timelines that consume raw data from external devices generating events.
- Timeline Event Types API allows you to construct and maintain event types that can be shown in an event timeline. Event types correspond to actions taken by organizations and customers. Event types can be grouped by category. This API also supports the ability to associate templates to specific event types.
- Timeline Configuration API includes methods that can constrain a very large timeline with views that read from the timeline service and inject context into the raw event data being presented. This API includes methods for creating event templates and rendering contexts for these views.

Structured and Unstructured Data

The SM Platform relies on a combination of structured and unstructured data storage to provide our clients with a high performance solution.

Structured data can be a timeline for a specific customer and likely contains several meaningful properties. Structured data has been optimized for display. The primary display is the activity log presentation in various SMP UI screens. These core applications perform transactions via SMP server-to-server APIs implemented in Ruby. These public APIs create and manage meaningful data constructs such as timelines, views, event types and event categories.

Unstructured data, on the other hand, is really just raw data with a timestamp and a timeline reference. It is payload data that has been built in both Java or Go and has been optimized for speed. This data is pulled from an SQL-based data store and can be accessed with an internal Timeline APIs written in Java.

The Timeline APIs provide ways of accessing structured data from an unstructured data store. The APIs offer critical services to many other APIs that need to access customer events, which are stored in timelines.

Events in Timelines

Events flow into timelines. They get written to a timeline at a variety of ingestion points. The Events API is an ingestion point. Others, for example, include APIs for transactions, offers and POS domains. Events can be associated with:

- Activity - Any predefined, customer-triggered event that indicates a customer has performed a particular behavior. Examples of basic activity events range from digital engagement with content, such as opening an app or clicking on a web form, to other simple activity events, such as abandoning a shopping cart within a mobile app or on an e-commerce site. May optionally include additional event attributes such as a count for the number of times the event has occurred and the date of its occurrence.
- Location - A customer-triggered event that indicates a location-specific activity such as a customer entering, leaving, staying at, or checking into a particular venue or set of venues. These events can enable the caller of the API to trigger location-specific content and messaging. Common metadata includes location-specific information such as the device location of the event (longitude, latitude), geofence crossed, venue and time of the event.
- Purchase - A customer-triggered event that indicates purchases such as a transaction value and, optionally, an item code or SKU. These events enable the caller of the API to trigger content and messaging based on SKU-level purchase events.
- State Change - Data reflecting customer state changes associated with events. Generally, these events are not static, but rather defined for customers relative to a point in time. For example, events might include using a loyalty card, submitting a survey, or celebrating a birthday. When events occur, state changes accompany them, such as changing from one tier to another or opting in/out of a loyalty program. There is often a boolean attribute flipping from one state to another.

Use Cases

These use cases are typical for the Timeline API:

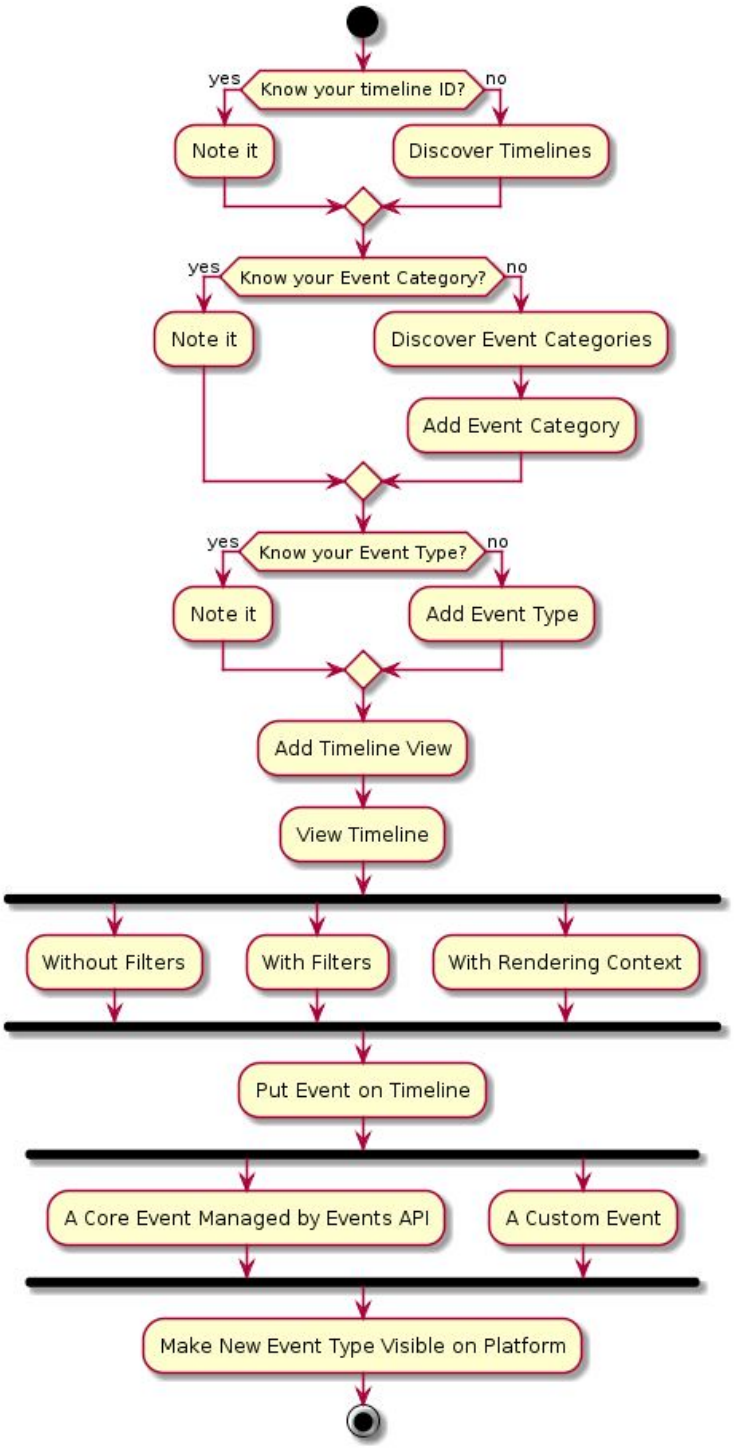
- Loyalty application to display user activity in a widget on a mobile application.
- Client web application to display user customer service notes in a preferences center application interface.

Data Configuration Prerequisites

This workflow requires that your customer success representative or integration engineer has ensured that the system has been configured. This process includes configuring log files to communicate with queues as well as setting up a few different timeline services.

Step by Step

The workflow for displaying specific data from a timeline follows the step-by-step decision tree diagrammed below:



The sections that follow reflect the decisions and actions detailed in this diagram. Note that throughout this workflow external IDs are featured in the specification of endpoints, but internal IDs can also be used.

When issuing curl commands for platform transactions, adhere to the following syntax:

- Begin each curl command with either `POST` or `GET`.
- Specify: `-H 'Content-Type: application/json' -H 'authorization: Basic AUTH_ID'`
- Begin URL with same endpoint + API key: `https://[ENDPOINT]/priv/v1/apps/API_KEY`

Step 1: Know the Timeline ID

By default, the timeline, or stream, shows all event data. The first step for defining the data you want to display from a timeline is knowing which timeline you want to work with. If you know the timeline, note its ID and proceed to Step 2. The timeline ID is used for subsequent queries and is dependant on environment.

If you don't know it, you need to discover it from among the system's existing timelines. Using the Timeline Service API, specify this endpoint to see which timelines exist in the system:

```
GET /v1/apps/:api_key/timelines/streams
```

After the endpoint makes the request for timelines, the platform returns a response object, which is shown below:

Response


```
{
      "status": "ok",
      "streams": [
            {
                  "stream": {
                        "created_at": "2017-07-28T15:52:49Z",
                        "disabled": false,
                        "event_stream_stream_type_id": 1,
                        "id": 14,
                        "organization_id": 13,
                        "updated_at": "2017-07-28T15:52:49Z"
                  }
            }
      ]
}
```

This response shows that the system has a single timeline (represented by the *stream* object) with an ID of *14*. For more information, see Retrieve All Timeline Services.

Now, having identified the single timeline in the system, you can view what event categories have been configured for the system and then determine whether or not you need to add a new one for the data you want to present.

Step 2: Know the Event Category

The second step for defining the data you want to display from a timeline is knowing the event categories of the event types you want to show. Note that the platform is pre-configured with a collection of event categories for several kinds of data, including points, offers, transactions and campaigns. So it's entirely possible that the category you want to display already exists.

If you know the event category, note its ID and proceed to Step 4. If you don't know it, you can discover what event categories already exist in the system. Without knowing the categories, subsequent API calls may create redundant data. Using the Timeline Event Types API, specify this endpoint to retrieve all event categories for all timelines, which, in this workflow, happens to be a single timeline:

```
GET /v1/apps/:api_key/timelines/categories
```

After the endpoint makes the request for categories, the platform returns a response object, which is shown below:

Response


```
{
      "status": "ok",
      "result": [
            {
                  "event_category": {
                        "created_at": "2017-07-28T15:52:49Z",
                        "description": null,
                        "id": 118,
                        "name": "POINTS",
                        "organization_id": 13,
                        "slug": "POINTS",
                        "updated_at": "2017-07-28T15:52:49Z"
                  }
            },

            .....Truncated...
      ]
}
```

This truncated response shows that the system has a single event category with a name of *POINTS*. For more information, see Retrieve Event Categories.

Now, having identified the existing event categories configured for the system, you can either note the one you want to use or add a new one.

Step 3: Add an Event Category

If you don't see the event category you need, you may need to create one. The third step for defining what data to show from a timeline is adding an event category for the kind of events you want to present. After verifying the categories in the previous step, you can ensure that the new category you create serves a new business or integration purpose.

Using the Timeline Event Types API, specify a new category in the endpoint with a category name. The new event category shown in the endpoint below is *Z_POINTS*:

```
POST /v1/apps/:api_key/timelines/categories?category[name]=Z_POINTS
```

After the endpoint makes the request for categories, the platform returns a response object, which is shown below:

Response


```
{
      "status": "ok",
      "event_category": {
            "updated_at": "2018-06-26T13:56:20Z",
            "created_at": "2018-06-26T13:56:20Z",
            "slug": "Z_POINTS",
            "id": "120",
            "name": "Z_POINTS"
      }
}
```


This response shows the addition of the *Z_POINTS* category, with an ID of *120*. For more information, see Create an Event Category. Note that event categories are not visible on their own. Each event type must be individually specified in a rendering context, a step discussed below.

Now, having identified or created a new event category, you can determine what event type to display within that event category.

Step 4: Know the Event Type

The fourth step for defining what data to present from a timeline is knowing or creating the event type you want to show.

Event types are business actions. Currently, there is no way to validate the existing business event types in the system. However, you do have two options for discovering an existing one:

- You can consult your customer success representative or integration engineer for assistance in identifying the system's existing event types.
- You can consult the following list of the platform's default categories/types:

Category	Types
CAMPAIGNS	EMAIL_OPENED PUSH_MESSAGE_OPENED VIDEO_WATCHED
LOYALTY	KEYWORD_ENTERED LOYALTY_CARD_LINKED LOYALTY_RULE_ACHIEVED REGISTRATION USER_TAG_DROPPED
NOTES	NOTE_ADDED
OFFERS	GIFT_REDEEMED OFFER_CLAIMED OFFER_REDEEMED
POINTS	POINTS_AWARDED POINTS_COMPED POINTS_EARNED POINTS_USED
PROMO_CODES	PROMO_CODE_RECEIVED PROMO_CODE_USED
PURCHASE	PURCHASE
PURCHASE_TRANSACTIONS	PURCHASE_TRANSACTION
TIERS	TIER_ADVANCED TIER_RESET TIER_TRANSIT

If you know the event type you intend to display data for, note it and proceed to Step 6. If you don't, you can add one according to the discussion below.

Step 5: Add the Event Type

If you don't know the event type, the fifth step for defining what data gets presented from the timeline is creating a new one. Using the Timeline Event Types API, specify this endpoint to create an event type for a timeline:

```
POST /v1/apps/:api_key/timelines/14/event-types
```

Note that the parameter *14* included in this endpoint is the timeline ID, which is shown as the *id* attribute of the *stream* object returned in the response featured in Step 1.

The endpoint passes in the *event_type* request object for a new event type called *PURCHASE*.

Request

```
{
    "event_type": {
        "description": "New Purchase Event",
        "event_stream_event_category_id": 120,
        "name": "PURCHASE"
    }
}
```

This request for the new event type contains an *event_stream_event_category_id*. Its value is set to *120* because that is the id associated with the *event_category* object named "Z_POINTS," which is returned in the response featured in Step 3. In effect, *event_stream_event_category_id* in this *event_type* request object maps to *id* of the *event_category* response object depicted in Step 3.

The platform then returns a response object:

Response

```
{
    "status": "ok",
    "event_type": {
        "description": "New Purchase Event",
        "event_stream_event_category_id": 120,
        "id": 1235,
        "event_stream_stream_id": 14,
```

```
        "slug": "PURCHASE",
        "name": "PURCHASE"
    }
}
```

This response shows the newly added event type *PURCHASE* with its corresponding event category (*event_stream_event_category_id*) of *120* and a timeline ID (*event_stream_stream_id*) of *14*. Note too that the new event type's *id* is *1235*. For more information, see Create an Event Type.

Now, with the event type discovered in Step 4 or created in Step 5, you can add a new timeline view.

Step 6: Add a New Timeline View

The sixth step for defining the data you want to present from a timeline is adding a new timeline view. The timeline view captures whatever subset of the timeline's entire event history that you want to present. It is the bare minimum that is required to call the timeline for a particular user. Using the Timeline Configuration API, specify this endpoint to create a new view for a timeline, which is represented in the endpoint as the *event_stream_stream_id*, an integer of *14*:

```
POST /v1/apps/:api_key/timelines/14/views
```

The endpoint passes in the *stream_view* request object for a view called *NEW_VIEW*. This view is filtered by *since* and *count* values, which are set in the *query* attribute, as shown below:

Request

```
{
      "stream_view": {
            "query": "since, count",
            "event_stream_stream_id": 14,
            "name": "NEW_VIEW",
            "description": "A new view on our data"
      }
}
```

The platform then returns a response object:

Response

```
{
      "status": "ok",
```

```
      "stream_view": {
            "description": "A new view on our data",
            "id": 59,
            "query": "since, count",
            "event_stream_stream_id": 14,
            "slug": "NEW_VIEW",
            "name": "NEW_VIEW"
      }
}
```

This response shows the newly added *stream_view* object with a name of *NEW_VIEW* and a corresponding *event_stream_stream_id* (timeline ID) of *14*. For more information, see Create a Timeline View.

At this point in the workflow, we have obtained or created a new event category and event type as well as a view of the timeline data that is filtered using the *since* and *count* attributes. For more information on these attributes, review the “Endpoints” section for Retrieve Events for a Single View.

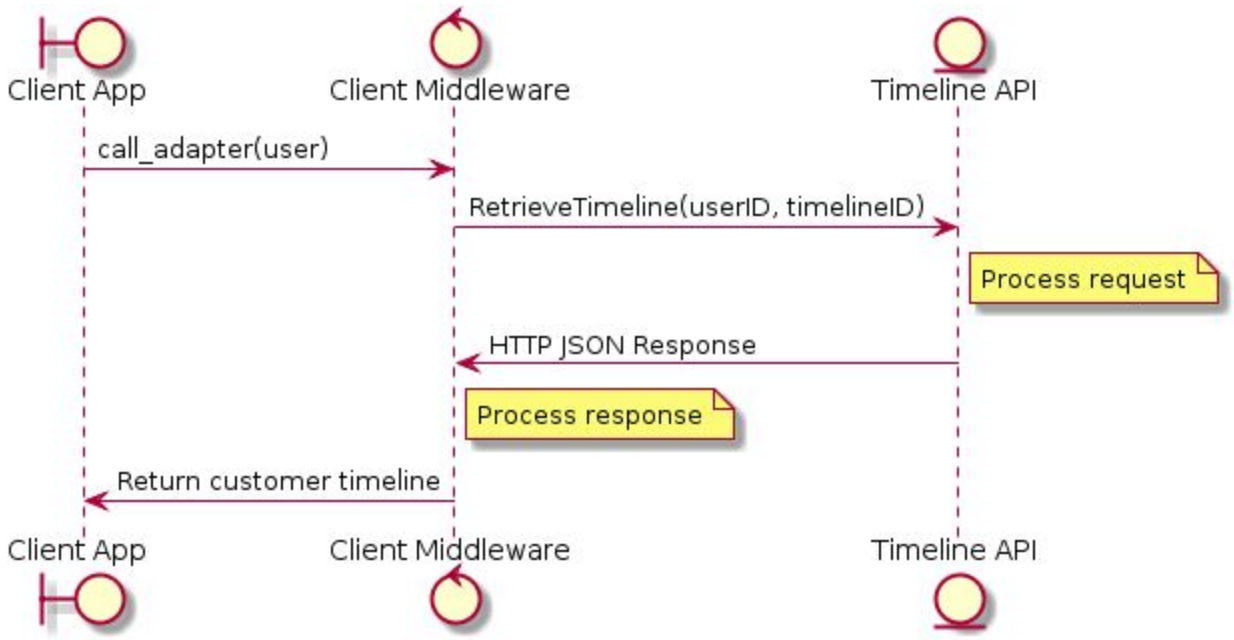
Step 7: View a Timeline

Now that you have defined the view of the timeline that you want, you can actually present it in this seventh step of the workflow. For example, you might be displaying customer activity in a mobile app or the SMP UI. Whatever the use case, the platform provides three different ways of showing a view:

- Without Filters
- With Filters
- With Rendering Context

Without Filters

When you view a timeline without any filters, the timeline displays the maximum number of events. With this view, you can pull the timeline and retrieve all data for the customer. The pattern below depicts a client app making a call to retrieve a complete customer timeline, without any data filtered out.



Use the Timeline Service API to specify this endpoint and retrieve and view a timeline without any filters:

```
GET
/v1/apps/:api_key/external/users/{{external_id}}/timelines/NEW_VIEW
```

After the endpoint makes the request for the entire, unfiltered timeline, the platform returns a response object, which is shown below:

Response


```
{
      "status": "ok",
      "result": [
            {
                  "target_id": 114910,
                  "event_stream_stream_id": "14",
                  "event_stream_event_type_id": 1235,
                  "event_stream_event_category_id": 120,
                  "timestamp": 1530219732000,
                  "created_at": 1531166534000,
                  "event_stream_payload": {
                        "parent_name": "Slug",
                        "country": "USA",
                        "event_type_slug": "PURCHASE",
                        "timezone": "+00:00",
                        "channel": "InStore",
```

```
                        "description": "",
                        "external_id": {
                              "key": "external_id",
                              "value": "zlwxIOdXjpLLr4zx",
                              "parent": "",
                              "type": "MAPPED"
                        },
                        "rewards_system_id": "6",
                        "player_id": {
                              "key": "player_id",
                              "value": "114910",
                              "parent": "",
                              "type": "MAPPED"
                        },
                        "model_type_name": "",
                        "event_category_name": "Z_POINTS",
                        "transaction_time": {
                              "key": "transaction_time",
                              "value": "1530219732.0",
                              "parent": "",
                              "type": "MAPPED"
                        },
                        "currency": "457",
                        "developer_id": "13",
                        "sub_channel": "Mobile",
                        "model_type_id": "",
                        "transaction_id": "RR6cyohSraGIxqG5",
                        "used_msr_reward": "false",
                        "amount": "2070.0",
                        "awarded_merchant_points": "",
                        "award_limit_reached": "",
                        "card_number": "",
                        "retailer": "",
                        "price_amount": "2070.0",
                        "store": "R2q",
                        "model_id": "",
                        "transaction_type": "Create",
                        "application_id": "145",
                        "event_type_name": "PURCHASE",
                        "override_price_amount": "",
                        "event_category_slug": "Z_POINTS",
                        "user_id": "8fd97dda-83b2-11e8-9e43-f75d1b1d641c",
                        "subtotal": "2070.0",
                        "qty": "1",
                        "name": "Hy1KVfajuL",
                        "discount_price_amount": "0",
                        "postal_code": "01875",
                        "request_id": "f80256c0-83b2-11e8-9fd8-6a371b1d641c"
                  },
                  "contexts": [
                        {
                        }
                  ]
```

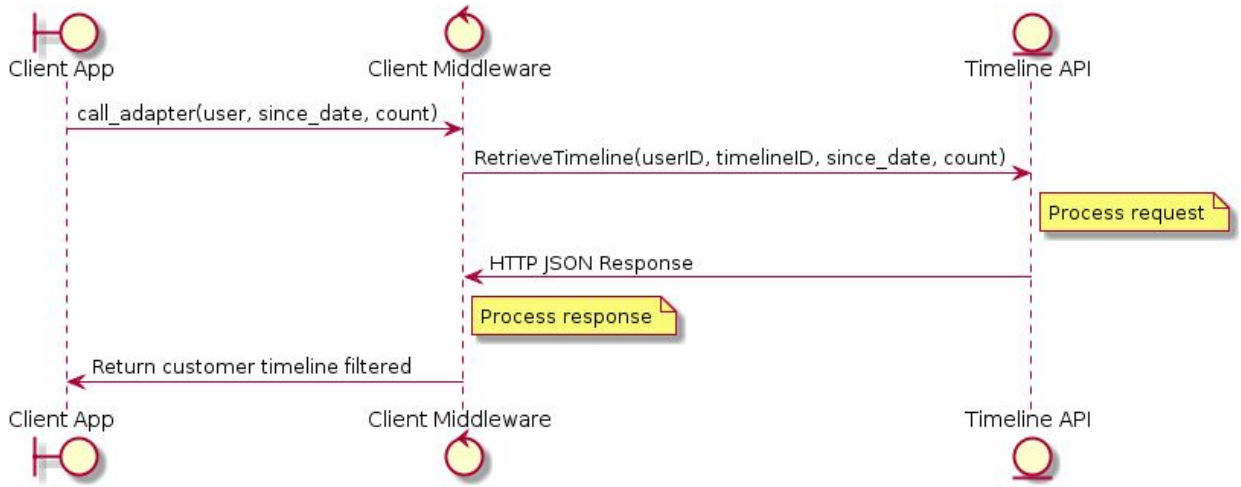
```
            },
            ...Truncated...

        ],
        "grouping_field": null
}
```

For more information, see Retrieve Events for a Single View.

If the timeline view without filters is not your preference, you can choose to view a timeline *with* filters, which is addressed below.

With Filters

When you view a timeline with a combination of filters, you reduce the number of events being shown in the view. The pattern below depicts a client app making a call to retrieve a customer timeline, filtering with *Since* and *Count* values.



Using the Timeline Service API, you can specify an endpoint that retrieves and views a timeline with several active filters. These filters can be passed as parameters in the endpoint URL. Here are the possible filter types, each with an example:

- Context slug: `context_slug=ABC`
- Since: `since=1484585770`
- Count: `count=2`
- Event Category ID: `filter[event_category_id]=123`
- Event Type IDs (multiple): `event_types=12,13,14`
- Target ID: Internal ID from SessionM

For more information on these parameters, see the "Endpoint Parameters" section of Retrieve Events for a Single View.

The following endpoint utilizes parameters for count, since, event types and event category:

```
GET
/v1/apps/:api_key/external/users/{{external_id}}/timelines/NEW_VIEW?count
=1
&since=1530045547&event_types=12,13,14&filter[event_category_id]=123
```

After the endpoint makes the request for the filtered timeline, the platform returns a response object similar to the one shown in Without Filters. Bear in mind that by adding filters to your GET, you reduce the number of events that appear in the response.

If the timeline view with or without filters is not your preference, you can choose to view a timeline with a rendering context, which is addressed below.

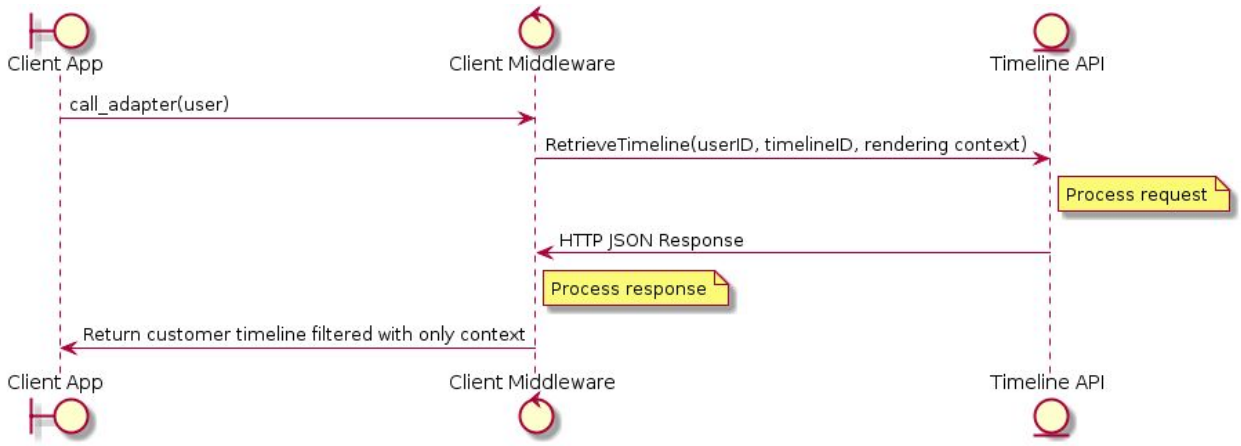
With Rendering Context

When you create a custom view with a rendering context, you avoid filtering a view with SessionM event type or event type category IDs. In this case, you can create a custom rendering context for your timeline using the Timeline Event Types and Timeline Configuration APIs in conjunction with the SessionM Platform UI.

A rendering context specifies a context for a view to be presented in an associated template. For example, you could define a rendering context for customers engaged in an iOS-based mobile experience. In addition, the context could be associated with an event template, along with its association to an event type. Once a rendering context has been created, it is not editable via any platform APIs; so if any mistakes are made, consult your SessionM customer success representative or integration engineer.

Note: A rendering context is somewhat like an inverse filter. Nothing is shown until you explicitly add a reference from an event type to the rendering context. A rendering context without any links returns no results.

The pattern below depicts a client app making a call to retrieve a customer timeline using a rendering context.



The following procedure guides you through the steps required to create a rendering context for a customer:

1. Retrieve the *event_type_id* by using your browser's inspector function to locate the corresponding *event_stream_event_type_id* in the activity feed for your customer. For more information, see Appendix A.

2. With the *event_type_id*, you can use the Retrieve Rendering Templates API to retrieve a rendering template ID, which is specified by *event_stream_event_template_id*.

3. Add a new timeline view as per Step 6: Add a New Timeline View. Note the ID - in this case *59* - shown in the *stream_view* response object. You can then specify this ID as the *event_stream_stream_view_id* parameter in the endpoint shown in step 5 of this procedure.

   Or, alternatively, you can get all of the necessary stream view details for an existing view by using the Retrieve Details on a Timeline View API.

4. Now create an *event_stream_rendering_context_id* by running a valid curl like this:

```
POST /v1/apps/:api_key/timelines/rendering_contexts
?rendering_context[name]=name_here
```

   The platform returns the following response:

Response
`{` `  "status": "ok",`

```
    "rendering_context": {
        "id": 56,
        "name": "api:event2",
        "slug": "API_EVENT2",
        "organization_id": 13
    }
}
```

5. Finally, with all the data collected in this procedure, you can add the *event_type* to the event stream rendering context. For example:

```
POST
/v1/apps/:api_key/timelines/{{stream_id}}/event-types/{{event_
type_id}}/templates?rendering_template[event_stream_event_temp
late_id]={{event_stream_event_template_id}}&rendering_template
[event_stream_stream_view_id]={{event_stream_stream_view_id}}&
rendering_template[event_stream_rendering_context_id]={{event_
stream_rendering_context_id}}&rendering_template[version]={{ve
rsion}}
```

The platform returns the following response:

Response

```
{
    "status": "ok",
    "rendering_template": {
        "id": 1158,
        "event_stream_event_template_id": 1227,
        "event_stream_stream_view_id": 59,
        "event_stream_rendering_context_id": 56,
        "version": 1
    }
}
```

6. Now, with your new rendering context defined, you can call it:

```
GET /v1/apps/:api_key/timelines/GROUPED_VIEW?context_slugs=name_here
```

The platform returns a response that is similar to what's returned for a timeline request in Without Filters.

Now that you have defined an event category and type, along with an associated view, it's time to generate some events on the timeline that can be displayed later in that view.

Step 8: Put an Event on the Timeline

Timelines record events. So, before you can see a particular view of timeline data, you need some actual events to see! Therefore, the eighth step of this workflow is putting an actual event on the timeline, one that can be displayed or rendered. You can add two kinds of events:

- Core Event Managed by Events API
- Custom Event

Core Event Managed by Events API

When you add a core event, you define it with attributes that make it available to the other modules on the platform. This metadata, or logic, makes them integral to other kinds of platform objects, such as campaigns and offers. Core events are also written to platform logs.

Using the Events API, specify this endpoint to create an event on the timeline:

```
POST /v1/apps/:api_key/external/users/{{external_id}}/events
```

The endpoint passes in the *events* request object for a purchase event for an amount of 1741, which is shown below:

Request

```
{
      "events": {
            "purchase": [
                  {
                        "parent_name": "Racer",
                        "price_amount": 1741,
                        "name": "YZSC9l7DNF",
                        "currency": "457",
                        "sub_channel": "Mobile",
                        "transaction_id": "coUxkGSFpTDu1nn0",
                        "qty": 1,
                        "item": "Racer",
                        "amount": 1741,
                        "postal_code": "01875",
                        "store": "cYG",
                        "time": 1529826439,
                        "transaction_type": "purchase",
                        "subtotal_amount": 1741,
                        "country": "USA",
```

```
                        "channel": "InStore"
                    }
                ]
        }
}
```

Note that one way data is defined in the request object is via the SMP Transactions domain API. Documentation for this API is anticipated for Q1 of 2019.

The platform returns a response object:

Response

```
{
        "status": "ok",
        "available_points": 610,
        "notifications": [

        ],
        "user": {
                "available_points": 610,
                "tier": "GOLD",
                "deltas": {
                        "available_points": 244
                }
        },
        "behaviors": {},
        "deltas": {}
}
```

This response shows a purchase made by a customer with *GOLD* tier status whose available points decreased by *244*, leaving *610* available points. For more information, see Create an Event for a Customer.

Alternatively, you may want to create a custom event that is not available to other platform modules. This custom event is discussed below.

Custom Event

When you create a custom event, you add it directly to the timeline, bypassing all processing from the other SessionM modules. This is a perfect solution for customer data that needs to be displayed in the platform but NOT processed by our system for campaigns, points, offers, etc. Using the Timeline Service API, specify this endpoint to add a custom event - not a core SessionM event - to a timeline:

```
POST
/v1/apps/:api_key/external/users/{{external_id}}/timelines/{{timeline_id
}}
```

The endpoint passes in the event request object, which is shown below:

Request

```
{
      "timestamp": "2017-06-05 12:30:00",
      "transactional": false,
      "stream_type": "USER",
      "event_type": "YPURCHASE",
      "payload": {
      }
}
```

The platform returns a response object:

Response

```
{
      "status": "ok",
      "result": {
            "saved": 121
      }
}
```

For more information on using this endpoint and its request object, see Publish an External Event into a Timeline.

With an event added to the timeline, you can now make data for the corresponding event type visible on the platform.

Step 9: Make New Event Type Visible on the Platform

One common use case for timeline data associated with an event type is making it visible in the SMP UI. This ninth step of the workflow makes the data for the new event type visible on the platform's activity log.

Before you begin, you may want to ensure that the activity log shown in the SMP is enabled. Consult your customer success representative or integration engineer to verify that the following settings are enabled in the Timeline Series Config tab of the Super Admin page, which is located in the Admin & Rights 2.0 Module:

- Stream view: `end_user_member_statement`
- Rendering context: `mmc_customer_campaigns`

To make your new event type appear in the SMP:

1. Using the Timeline Configuration API, define an event template for the event type with following endpoint and request object:

```
POST /v1/apps/:api_key/timelines/event_templates
```

Request

```
{
    "event_template": {
        "html": "<p>{{event_category_slug}} {{storenumber}}
{{event_type_slug}}</p>",
        "plain_text": "plain text",
        "event_stream_event_type_id": 46
    }
}
```

The platform returns the following response:

Response

```
{
    "status": "ok",
    "event_template": {
        "html": "<p>{{event_category_slug}} {{storenumber}}
{{event_type_slug}}</p>",
        "plain_text": "plain text",
        "id": 46,
        "event_stream_event_type_id": 46
    }
}
```

2. Using the Timeline Event Types API, assign this new event template to the rendering context and view it as a rendering template with this endpoint and request object:

```
POST
/v1/apps/:api_key/timelines/[STREAM_ID]/event-types/[EVENT_TYP
E_ID]/templates
```

Request

```
{
    "rendering_template": {
        "event_stream_stream_view_id": 1,
        "version": 1,
        "event_stream_event_template_id": 46,
        "event_stream_rendering_context_id": 1
    }
}
```

The platform returns the following response:

Response

```
{
    "status": "ok",
    "rendering_template": {
        "event_stream_stream_view_id": 1,
        "event_stream_rendering_context_id": 1,
        "id": 46,
        "version": 1,
        "event_stream_event_template_id": 46
    }
}
```

3. If some of your event types have not displayed on the SMP, you can have your integration engineer run the following query to find missing event templates and/or rendering templates.

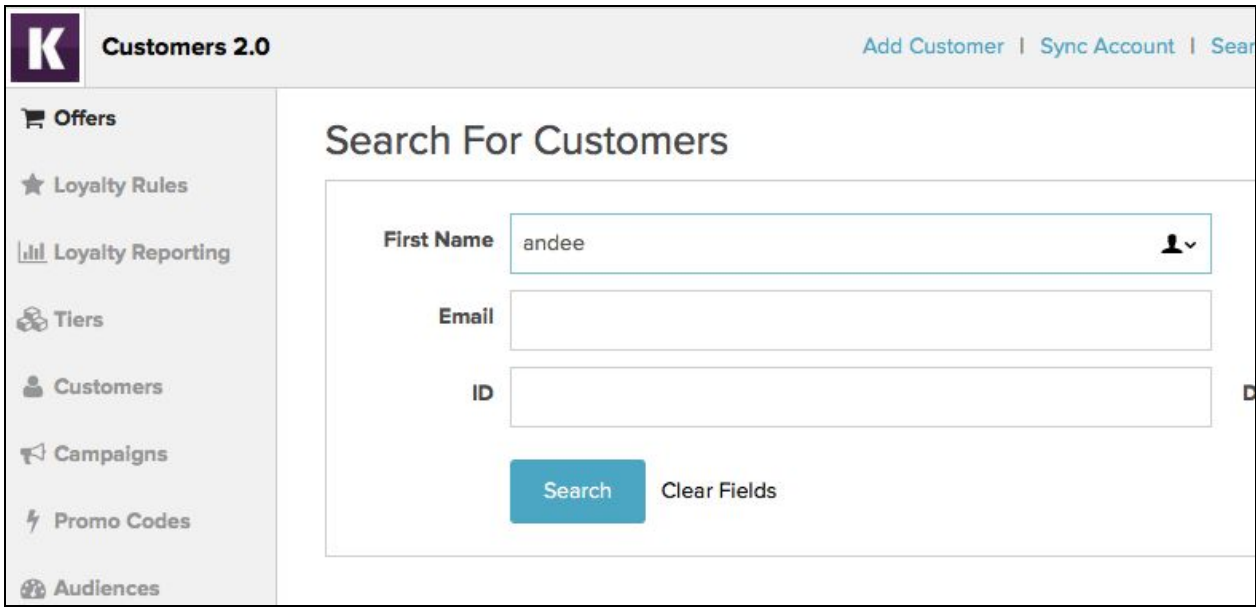
```
SELECT typ.id event_type_id, typ.name, cat.id, cat.name,
tmpl.id event_template_id, tmpl.html, rt.id
rending_template_id, sv.slug view_slug, rc.slug
rendering_context_slug
FROM event_stream_event_types typ
inner join event_stream_event_categories cat on cat.id =
typ.event_stream_event_category_id
left outer join event_stream_event_templates tmpl on
tmpl.event_stream_event_type_id = typ.id
left outer join event_stream_rendering_templates rt on
rt.event_stream_event_template_id = tmpl.id
left outer join event_stream_stream_views sv on sv.id =
rt.event_stream_stream_view_id
```

```
left outer join event_stream_rendering_contexts rc on rc.id =
rt.event_stream_rendering_context_id
```

Appendix A: Accessing Timeline Data from a Browser

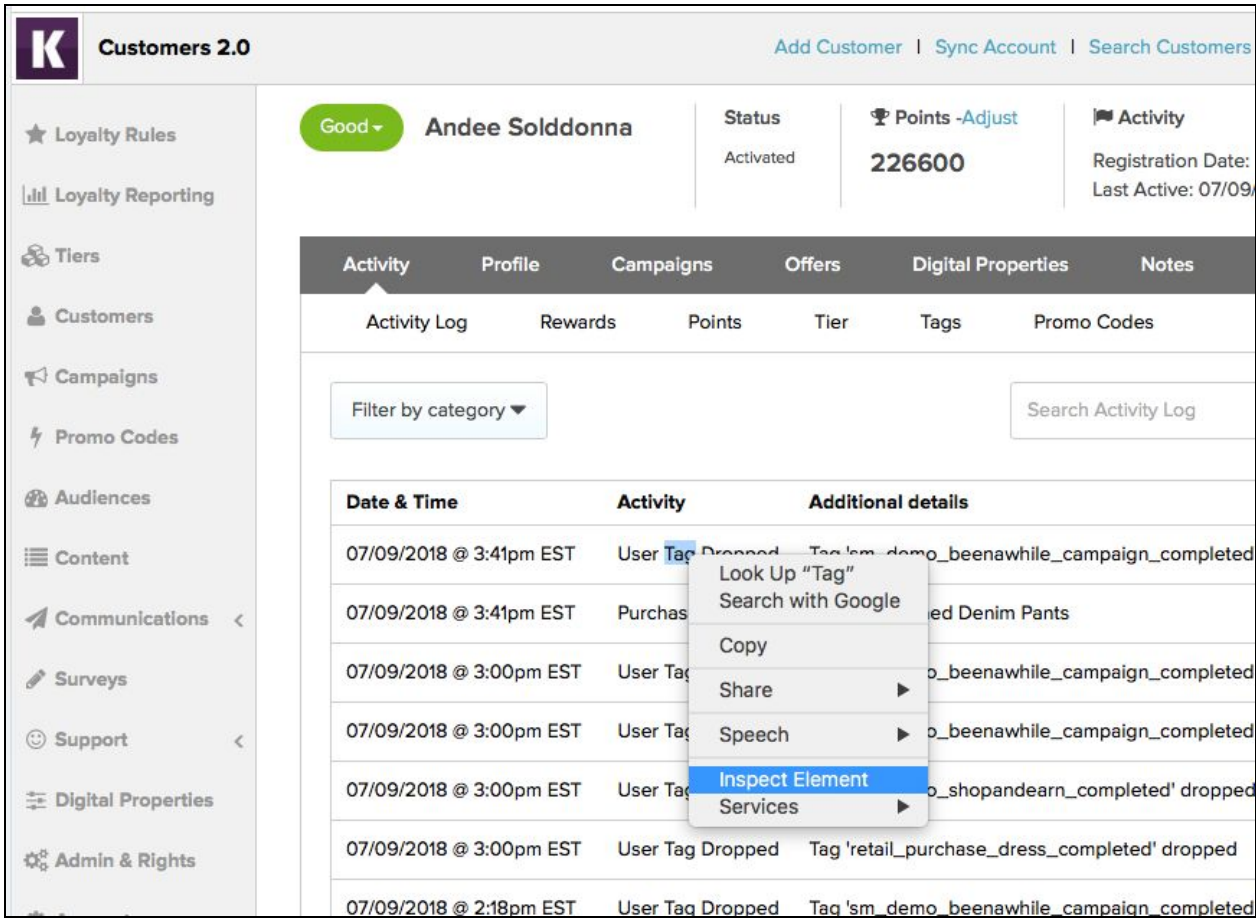
Once you have completed the timeline workflow, you can access and view timeline data from a browser by performing the following steps:

1. In the Customers Module of the SMP UI, use the Search For Customers dialog to locate the customer profile that has the event you want to target:



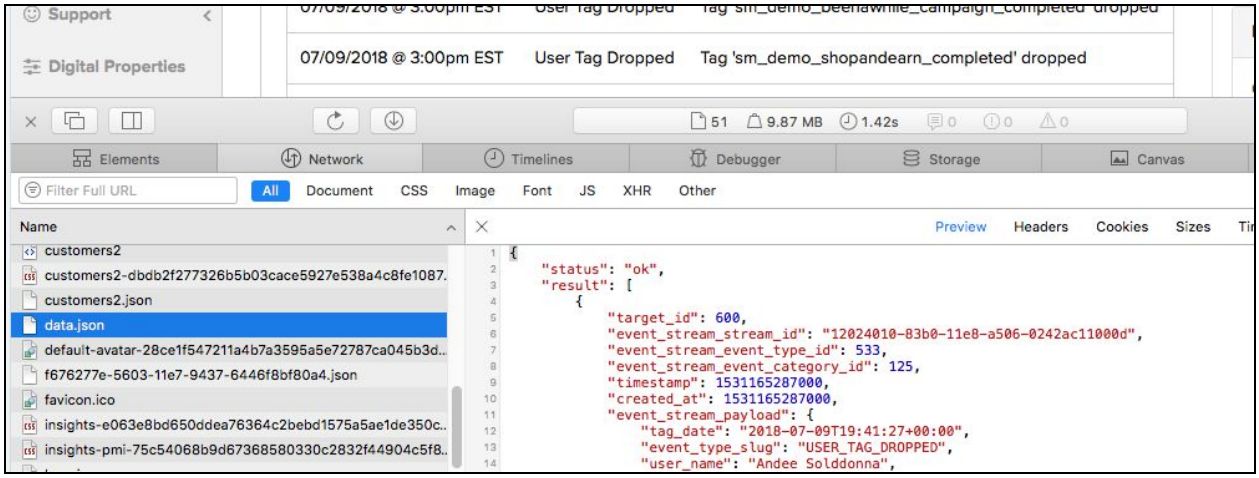
   The customer profile opens with the Activity Log displayed.

2. In the Activity Log table, right click any entry and, from the list of options, pick Inspect Element (in Safari) or Inspect (in Google Chrome), as shown below:



A new inspect page opens in the browser.

3. Go to the Network tab and refresh the page. A file called *data.json* will appear containing the activity stream in JSON format, as shown below:



As shown in the JSON file, the event type ID is *event_stream_event_type_id*; the event category ID is *event_stream_event_category_id* and so on.

Note that should errors occur as you perform the steps in this procedure, it can mean that your timeline services are set up incorrectly. For more information, consult your customer success representative or integration engineer.