Java Script SDK

Manuale di configurazione:

Versione 1.0.0

21 maggio 2019

Predixit S.r.l. Società a Responsabilità Limitata

Sede legale: Milano Piazza Santissima Trinità 9, CAP 20154

C.F.e Partita IVA: 09139100961

Cap. sociale euro 196.359,00 i.v.

Questo "Manuale" contiene informazioni e dati confidenziali riguardanti ***Predixit S.r.l.*** *("****Predixit****" o "l'Impresa"). Accettando questo memorandum, il ricevente si impegna a verificare che i propri direttori, funzionari, dipendenti, e rappresentanti usufruiscano di tale memorandum e delle informazioni in esso contenute esclusivamente per valutare questa specifica operazione di finanziamento con l'Impresa, e per nessun altro scopo. Il ricevente si impegna a non divulgare tali informazioni ad una terza parte, e si impegna a riconsegnare all'impresa questo memorandum, tutte le sue eventuali copie e qualsiasi altro dato connesso con esse, nel momento in cui ciò dovesse essere richiesto dall'impresa.*

I dati contenuti in questo memorandum sono stati ottenuti dall'impresa e da altre fonti. Qualsiasi stima o proiezione qui presente è risultato dell'analisi effettuata sull'impresa, la quale potrebbe essere o non essere corretta. L'Impresa non fornisce garanzia, esplicita o implicita, che quanto riportato nel seguente documento rappresenti con precisione e/o completezza le informazioni riguardanti l'impresa e niente nel seguente documento deve essere inteso come fedele rappresentazione di situazioni e/o dati passati e/o futuri dell'impresa analizzata. Questo memorandum non si prefigge l'obiettivo di contenere tutti i dati e le informazioni che potrebbero essere necessarie per portare a compimento tale operazione di finanziamento, e viene fatto esplicito invito al ricevente di effettuare in ogni caso la propria indipendente analisi dell'impresa e dei dati qui di seguito riportati.

Sommario

Predixit JavaScript SDK

Versione 1.0.0

La libreria *predixit-sdk.js* è una libreria JavaScript utilizzata per misurare come gli utenti interagiscono col vostro sito di e-commerce.

Questo documento fornisce una descrizione dei processi di integrazione richiesti per utilizzare la libreria all'interno del vostro sito e adottare i sistemi di Predixit per offrire ai vostri clienti un'esperienza d'acquisto personalizzata.

Panoramica

La libreria *predixit-sdk.js* di Predixit (da qui in avanti **SDK**), è una libreria JavaScript compatibile ed integrabile con qualsiasi sito di e-commerce, che, dopo alcune semplici azioni di configurazione, permetterà di tracciare il comportamento di clienti e visitatori all'interno del proprio sito di e-commerce, per costruire una profilazione dettagliata dei vostri utenti e personalizzare la customer experience tramite gli strumenti di analisi predittiva, raccomandazione e promozione di Predixit.

Il tracciamento del compotamento degli utenti sul vostro sito consentirà ad esempio di monitorare il numero e il tipo di interazioni col catalogo del vostro e-commerce durante l'esperienza di navigazione dell'utente: azioni quali impression, click e visualizzazioni su categorie e prodotti, visita a pagine di dettaglio, aggiunte a carrello e tutto quanto riguarda il funnel di acquisto, potranno essere tracciate e utilizzate per costruire un profilo completo dei vostri utenti.

Con SDK potrete realizzare tutto questo in maniera semplice: in un tipico sito di e-commerce, quando un utente clicca il pulsante "acquista" sul proprio browser, le informazioni riguardo l'acquisto vengono inviate al server che completa la transazione. In caso di successo, il server ridirige l'utente a una "thank you page", coi dettagli della transazione appena completata. Potete usare ad esempio in questa pagina la libreria *predixit-sdk.js* per inviare i dati circa la transazione appena completata alla piattaforma Predixit.

Allo stato attuale, gli eventi tracciabili tramite SDK sono:

- *Visualizzazione di una categoria;*
- *Click su un prodotto;*

- *Visualizzazione di un prodotto;*
- *Aggiornamento del carrello (aggiunta, modifica o rimozione di prodotti);*
- *Aggiornamento dello stato dell'ordine (checkout, completamento, cancellazione, eccetera);*
- *Impression (il numero di volte che un prodotto viene visualizzato in una lista, banner o componente all'interno del sito);*

Quali altri dati vengono tracciati da SDK?

Aggiungendo SDK al vostro sito, oltre alle azioni strettamente legate all'e-commerce viste in precedenza, verranno inviate a Predixit anche informazioni quali ad esempio:

- *il tempo totale che un utente spende sul vostro sito*
- *che pagine ha visitato all'interno del vostro sito*

In aggiunta l'indirizzo IP, lo user agent e la lingua dell'utente possono essere utilizzati per determinare:

- *la lingua in cui proporre i contenuti per un'esperienza di acquisto personalizzata*
- *la posizione geografica dell'utente*
- *quale browser e sistema operativo sta utilizzando*
- *il sito di origine*

Inclusione di SDK

Per poter utilizzare la libreria SDK nel proprio sito di e-commerce, è necessario includere quanto più "in alto" possibile nella sezione <head> di tutte le pagine del proprio sito il seguente codice JavaScript:

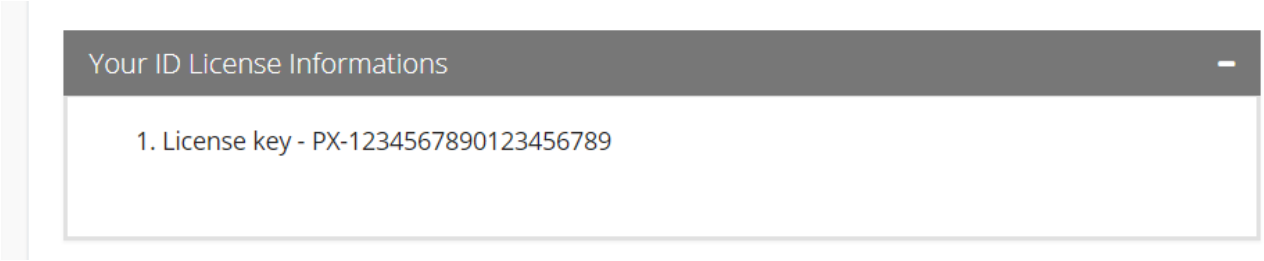
```
<script type="text/javascript" src="https://cdn.predix.it/sdk/dist/predixit-sdk.min.js" async=""></script>
```

Una volta fatto questo, al caricamento del sito, tramite il DOM della pagina si potrà accedere alle funzionalità di SDK attraverso l'oggetto globale **"pdxTrk"**.

Inizializzazione

L'unica azione obbligatoria per poter iniziare ad utilizzare la libreria è l'inizializzazione della stessa con la propria "**license key**".

Potete trovare la vostra "**license key**" nel tab "**Setup**" visibile nella pagina di modifica del proprio store nella "**Dashboard Predixit**", nella sezione "**Your ID License Informations**".



Per eseguire l'inizializzazione, subito dopo l'inclusione ed il caricamento della libreria come descritto sopra, si dovrà inserire la seguente istruzione, avendo cura di utilizzare il corretto valore di "**license key**":

```
pdxTrk.init('PX-1234567890123456789');
```

Per non dover effettuare controlli espliciti sulla conslusione del caricamento della libreria, si consiglia di unire le due istruzioni sopra (Inclusione e Inizializzazione) in un'unica istruzione come segue:

```
<script type="text/javascript" src="https://cdn.predix.it/sdk/dist/predixit-sdk.min.js" async="true" onload="pdxTrk.init('PX-1234567890123456789');"></script>
```

Sandbox

Per facilitare le operazioni di integrazione di SDK col proprio sito, Predixit mette a disposizione degli sviluppatori un ambiente di test speculare a quelli di produzione, dove poter effettuare prove e verifiche, senza preoccuparsi delle conseguenze o avere alcun impatto sui sistemi di produzione.

Si noti che **NESSUNA** azione tracciata tramite le API di Sandbox produrrà dei profili utenti all'interno della piattaforma Predixit, né potrà essere utilizzata per personalizzare l'esperienza di acquisto con prodotti reali presenti sul proprio sito di e-commerce.

Per utilizzare gli ambienti di Sandbox invece che quelli di produzione è necessario includere questo snippet al posto di quanto indicato in precedenza:

```
<script type="text/javascript" src="https://cdn.predix.it/sdk/dist/predixit-sandbox-sdk.min.js"
```

e utilizzare il token che vi è stato fornito assieme a questo documento (in genere nella forma "**SX**-*nnnnnnnnnnnnnnnnnn*").

```
pdxTrk.init('SX-1234567890123456789');
```

Debug mode

In fase di sviluppo e integrazione, si possono avere informazioni utili per il debug sulla console del browser utilizzato, ma di default il debug mode è abilitato esclusivamente per la piattaforma Sandbox.

Se si vuole abilitare/disabilitare il debug una volta inclusa la libreria si può utilizzare il metodo "**debugMode**" che accetta 2 parametri.

1. Il primo è un boolean che abilita e disabilita il debug;
2. Il secondo è un parametro opzionale che permette di settare un livello di debug, escludendo alcuni messaggi dai log in base alla tipologia;

   Si possono ottenere i livelli di debug disponibili attraverso il metodo "**getLoggerLevels**";

```
pdxTrk.getLoggerLevels();
```

Il quale fornirà i seguenti livelli:

```
{
    debug: 0,
    warning: 1,
    error: 2,
    info: 3
}
```

Ad esempio, per abilitare il debug con la sola visualizzazione degli errori e delle info si dovrà richiamare il metodo "**debugMode**" come segue:

```
pdxTrk.debugMode(true, pdxTrk.getLoggerLevels().error);
```

Per semplicità si potrà anche inserire direttamente il valore intero del livello di debug:

```
pdxTrk.debugMode(true, 2);
```

A questo punto, nella console del browser si potranno trovare alcune informazioni riguardanti la libreria, tutte precedute dalla chiave "**PdxTrk >>**" che permette di filtrare i messaggi per una più facile lettura.

Lingua utente

Per proporre ai propri utenti un'esperienza d'acquisto personalizzata, può essere necessario informare Predixit della lingua in cui l'utente sta navigando all'interno del sito di e-commerce.

Si pensi ad esempio all'erogazione di un componente di raccomandazione prodotti o a un pop-up che informa l'utente di una promozione in corso: queste informazioni richiedono una corretta localizzazione se erogati da un sito multilingua.

Per impostare la lingua con cui l'utente sta navigando nel sito è sufficiente richiamare il metodo "**setLang**", passando come parametro la lingua utilizzata seguendo indifferentemente la direttiva ISO639-1 con codice a 2 lettere (es. "en", "it", eccetera), o in alternativa la notazione col sub-code per la variazione locale (es. "en_US", "it_IT", eccetera).

L'esempio seguente imposta la lingua inglese:

```
pdxTrk.setLang('en');
```

La lingua selezionata viene memorizzata in un cookie, quindi essa verrà utilizzata automaticamente nelle interazioni successive, senza rendere necessario alcun intervento ulteriore da parte dello sviluppatore.

In caso non venga specificata nessuna lingua, il sistema assumerà che l'utente stia navigando nella lingua di default dello store.

Implementazione

SDK definisce diversi modelli di implementazione per le funzionalità offerte dalla libreria: per motivi di sviluppo e debug potrebbe ad esempio essere conveniente cambiare l'implementazione della libreria durante il suo utilizzo, adottando temporaneamente una versione "**mock**" che non effettui chiamate reali al server, in modo da verificare unicamente il comportamento del proprio client.

Questa funzionalità è resa disponibile dal metodo "**setImplementation**", il quale accetta in ingresso una chiave che identifica una delle implementazioni disponibili.

Le implementazioni disponibili si possono ottenere tramite il metodo "**getImplementations**"

```
pdxTrk. getImplementations();
```

Il quale fornirà le seguenti alternative:

```
{
    default: "default",  // actual implementation, send events to server
    mock: "mock",  // mocked implementation, do not send events to server
    optOut: "optOut" // simulate a user who opted-out, do not send events to server
}
```

Ad esempio, per utilizzare l'implementazione del tracker "**mock**" che non effettua chiamate al server si dovrà richiamare il metodo "**setImplementation**" come segue:

```
pdxTrk.setImplementation(pdxTrk.getImplementations().mock);
```

Per semplicità si potrà anche inserire direttamente la stringa corrispondente all'implementazione scelta:

```
pdxTrk.setImplementation ('mock');
```

Opt-out

In alcuni casi, potrebbe essere necessario disabilitare l'invio di dati a Predixit, senza rimuovere lo snippet JavaScript di inclusione dal proprio sito.

Ad esempio, tutto ciò potrebbe essere necessario se per motivi di privacy il vostro sito offre all'utente la possibilità di negare il consenso alla condivisione dei propri dati di navigazione con altri soggetti: ciò implica che SDK potrebbe inviare dati solo per alcuni utenti e non per altri.

Per disabilitare l'invio dei dati a Predixit è necessario utilizzare il metodo "**setOptOut**" come segue:

```
pdxTrk.setOptOut(true);
```

Il metodo “setOptOut” accetta un valore booleano true/false che permette di inibire/abilitare la raccolta delle informazioni ed il loro invio al server Predixit per questo specifico utente.

Le informazioni riguardanti l’opt-out verranno registrate in un cookie: sarà quindi necessario eseguire l’operazione un’unica volta, probabilmente a fronte del rifiuto del consenso da parte dell’utente.

Configurazione pool interval

Per il tracciamento di alcuni eventi, per questioni prestazionali può essere preferibile accumulare una molteplicità di ricorrenze in una coda, per poi inviare eventi tutti insieme con cadenza periodica.

Un tipico caso è rappresentato dal tracciamento delle impression dei prodotti.

SDK utilizza una configurazione per indicare la frequenza di “scarico” della coda: questo valore, impostato di default a 1 secondo, può essere modificato tramite il metodo “**setPoolInterval**”, il quale accetta in ingresso un valore in millisecondi che andrà a sovrascrivere il tempo di attesa che si vuole utilizzare.

Di seguito è proposto un esempio tramite il quale si setta l’intervallo a 2 secondi

```
pdxTrk.setPoolInterval(2000);
```

Si presti attenzione al fatto che la distruzione della pagina a seguito di cambio o chiusura della stessa potrebbe risultare nel non corretto scarico degli eventi nella coda: si suggerisce quindi di impostare un valore del parametro ragionevolmente basso per non rischiare di perdere eventi da tracciare.

Allo stato attuale, solo gli eventi di tipo IMPRESSION si avvalgono di questo meccanismo.

Tracciamento eventi

Come brevemente descritto in precedenza, SDK permette il tracciamento di una serie di eventi: si tenga presente che SDK invia dati alla piattaforma Predixit in maniera assolutamente asincrona e completamente trasparente all’utente, con tempi di risposta che si attestano su poche decine di millisecondi.

L’inclusione di SDK nel proprio sito non ha alcun impatto sulle performance del proprio sito o altera in alcun modo l’esperienza di navigazione dell’utente.

È possibile tracciare eventi attraverso il metodo “**trackEvent**”, che accetta in ingresso 4 parametri, dei quali solo i primi 2 sono obbligatori:

```
pdxTrk.trackEvent(eventType: Events, eventData: EventData, beaconMode?: boolean, waitTime?: number);
```

PARAMETRI

I parametri del metodo sono:

- **Tipo evento (*eventType*)**

La lista degli eventi disponibile si può ottenere attraverso il metodo "**getTrackEvents**"

```
pdxTrk. getTrackEvents();
```

Il quale fornirà le seguenti opzioni:

```
{
    productView: "PRODUCT_VIEW",
    productClick: "PRODUCT_CLICK",
    categoryView: "CATEGORY_VIEW",
    orderConfirm: "ORDER_CONFIRM",
    updateCart: "UPDATE_CART",
    impression: "IMPRESSION"
}
```

- **Dati dell'evento (*eventData*)**

Il tracciamento di una tipologia di evento necessita dell'invio di informazioni differenti in base alla tipologia stessa.

Per questo motivo si è reso disponibile un metodo che fornisce la struttura dei dati, che andrà popolata con le informazioni corrette in base alla tipologia di evento in esame.

Questo metodo si chiama "**getEventDataStructure**" e accetta in ingresso la tipologia dell'evento.

Ad esempio, se si volesse ottenere la struttura dei dati da popolare per l'evento "PRODUCT_VIEW", si dovrebbe eseguire la seguente istruzione:

```
pdxTrk. getEventDataStructure(pdxTrk. getTrackEvents().productView);
```

La quale fornirà le seguenti alternative:

```
{
    eventEntityId: null,
    origin: null
}
```

Come nei casi precedenti, per semplicità si può anche inserire direttamente la stringa corrispondente alla tipologia di evento scelto:

```
pdxTrk. getEventDataStructure('PRODUCT_VIEW');
```

Nel seguito verranno dettagliati i parametri specifici per ciascun evento.

- **Chiamata in modalità beacon – opzionale (*beaconMode*)**

Di default, le chiamate di tracking al server vengono eseguite tramite il metodo "**navigator.sendBeacon**". Questa tecnologia non è supportata da alcuni browser e per questo motivo, in modo assolutamente automatico e trasparente allo sviluppatore, nel caso in cui la modalità non sia supportata, la chiamata verrà eseguita in modalità "**XHR**".

Se si vuole disabilitare la chiamata in modalità beacon anche per i browser che supportano tale funzionalità, si deve impostare il valore di questo parametro booleano a false, ma in questo caso si prega di prestare attenzione particolare al prossimo parametro.

- **Tempo di attesa per la chiamata – opzionale (*waitTime*)**

Come si vedrà nel seguito, è molto importante intercettare un evento di click per inviare informazioni corrette a Predixit: è infatti assai comune registrare un'interazione che porterà l'utente verso una nuova pagina.

Alcuni browser (ad esempio Safari e Firefox) interrompono l'esecuzione di un JavaScript non appena viene avviato l'unloading della pagina (innescato ad esempio da un click dell'utente per seguire un link), il ché implica che SDK potrebbe non riuscire a inviare correttamente il tracciamento dell'evento se utilizzasse la modalità XHR.

Per ovviare a questo problema, dipendente dal browser utilizzato dall'utente, è possibile impostare un tempo di attesa che permetta la conclusione dell'invio dell'evento di tracciamento, prima della distruzione e cambio della pagina*: il valore è da intendersi come tempo di attesa *massimo* prima di effettuare il cambio pagina. SDK tratterà in autonomia l'evento, senza alcuna necessità di intervenire per lo sviluppatore.

Il valore va inserito in millisecondi: un valore accettabile normalmente è di **200**.

Attenzione, questo parametro sarà preso in considerazione solo ed esclusivamente nel caso in cui si sia impostato il parametro precedente a false

L'invio con la modalità beacon non soffre di questi problemi, pertanto il parametro viene ignorato se impostato.

Eventi

Di seguito sono riportati i dettagli per le varie chiamate di tracciamento con degli esempi di utilizzo.

CATEGORY VIEW:

Consente di tracciare la visualizzazione di una pagina di categoria da parte dell'utente.

Il metodo "trackEvent" dovrà essere chiamato con il parametro "CATEGORY_VIEW" come segue:

```
pdxTrk.trackEvent('CATEGORY_VIEW', eventData);
```

Dove

```
var eventData = {
        eventEntityId: 456,              // identificativo della categoria
        origin: 'HOME_PAGE',             // componente che ha originato l'evento*1
}
```

1 L'evento può scaturire da un determinato componente, ad esempio una sezione contenente i prodotti in evidenza nella home, una lista prodotti, un banner posizionato nell'header del sito o un componente di raccomandazione di Predixit, eccetera.
Il parametro "origin" ci permette di comunicare con una stringa libera a quale componente ci si riferisce: è molto importante che il parametro sia impostato correttamente, in modo da poter valutare le performance dei vari componenti che compongono il proprio sito.

PRODUCT CLICK:

Consente di registrare il click effettuato su un link di prodotto per accedere alla pagina di dettaglio: è importante che l'evento sia intercettato su tutte le ancore che portano alla pagina di dettaglio di un prodotto (es. il titolo del prodotto, l'immagine, etc.).

Il metodo "trackEvent" dovrà essere chiamato con il parametro "PRODUCT_CLICK" come segue:

```
pdxTrk.trackEvent('PRODUCT_CLICK', eventData);
```

Dove

```
var eventData = {
    eventEntityId: 123,                  // identificativo del prodotto
    origin: 'RECOMMENDATION_ BLOCK', // componente che ha originato l'evento
    campaign: 'CAMPAIGN_2232'    // valorizzare solo se il click è generato da un componente di Predixit*1
}
```

1 L'evento può scaturire da un componente di raccomandazione di Predixit. In questo ci basterà impostare il campo "campaign" col nome della campagna che ha generato l'evento e "origin" con il nome del componente, per poter misurare i risultati ottenuti da Predixit: tali informazioni sono reperibili nell'html restituito dalle API di personalizzazione di Predixit, si veda nel seguito per il dettaglio.

VISUALIZZAZIONE PRODOTTO:

Il metodo registra l'accesso dell'utente alla pagina di dettaglio di un prodotto.

Il metodo "trackEvent" dovrà essere chiamato con il parametro "PRODUCT_VIEW" come segue:

```
pdxTrk.trackEvent('PRODUCT_VIEW', eventData);
```

Dove

```
var eventData = {
```

```
    eventEntityId: 123,          // identificativo del prodotto
    origin: 'PRODUCT_LIST',      // componente che ha originato l'evento*1
}
```

1 Se a generare l'evento è stato direttamente un click, sarà SDK a utilizzare correttamente l'origin registrata: valorizzare il campo solo in caso di atterraggio sulla pagina in altro modo (es. referer, search engine, e-mail, eccetera).

UPDATE CART:

Il metodo è utilizzato per qualsiasi interazione col carrello: verranno gestite aggiunte, modifiche o cancellazioni dei prodotti al carrello.

Il metodo "trackEvent" dovrà essere chiamato con il parametro "UPDATE_CART" come segue:

```
pdxTrk.trackEvent('UPDATE_CART', eventData);
```

Dove

```
var eventData = {
    transactionId: 'a1b2c3',     // identificativo della transazione in corso
    delta: false,                // modalità di aggiornamento
    origin: 'PRODUCT_LIST',      // componente che ha originato l'evento*1
    items: [
        {
            itemId: 123,         // identificativo del primo prodotto
            quantity: 2          // quantità del primo prodotto
        },
        {
            itemId: 789,         // identificativo del secondo prodotto
            quantity: 1          // quantità del secondo prodotto
        }
    ]
}
```

1 Se a generare l'evento è stato direttamente un click, sarà SDK a utilizzare correttamente l'origin registrata: valorizzare il campo solo in caso di atterraggio sulla pagina in altro modo (es. referer, search engine, e-mail, eccetera).

Il campo "**delta**" del JSON, ci permette di utilizzare questa API in 2 modi distinti.

- **delta = true** indica una modifica relativa delle informazioni riguardanti il carrello.

  Ad esempio, con il JSON seguente:

```
var eventData = {
        transactionId: 'a1b2c3',
        delta: true,                    // modalità di aggiornamento relativa
        origin: 'PRODUCT_LIST',         // componente che ha originato l'evento
        items: [
            {
                itemId: 123,
                quantity: 2
            },
            {
                itemId: 456,
                quantity: -1
            }
        ]
}
```

  Si sta comunicando l'aggiunta al carrello del prodotto *123* in quantità *2* (se ce ne fosse già uno verrebbero sommati e otterremo la presenza del prodotto *123* in quantità 3 a carrello) e la rimozione (segno negativo) in quantità *1* del prodotto *456* (se ce ne fossero 2 già presenti, se ne sottrarrebbe 1, giungendo a 1).

- **delta = false** indica una modifica assoluta delle informazioni del carrello.

  Ad esempio, con il JSON seguente:

```
var eventData = {
        transactionId: 'a1b2c3',
        delta: false,                   // modalità di aggiornamento assoluta
        origin: 'PRODUCT_LIST',         // componente che ha originato l'evento
        items: [
            {
                itemId: 123,
```

```
            quantity: 2
        }
    ]
}
```

Corrisponde a comunicare che attualmente sul carrello c'è il prodotto *123* in quantità 2, indipendentemente dagli eventi inviati in precedenza.

Come conseguenza, utilizzare "delta: false" in combinazione con "quantity: 0" implica di fatto la rimozione di un prodotto dal carrello.

ORDER CONFIRM:

Il metodo consente il tracciamento del cambio di stato di un ordine, registrando ad esempio azioni legate all'avvio del processo di checkout, di completamento o di cancellazione di un ordine.

Il metodo "trackEvent" dovrà essere chiamato con il parametro "ORDER_CONFIRM" come segue:

```
pdxTrk.trackEvent('ORDER_CONFIRM', eventData);
```

Dove

```
var eventData = {
    couponCode: 'SUMMER_2019',  // eventuale codice del coupon utilizzato
    discountAmount: -5.00,      // eventuale sconto applicato (numero negativo)
    orderCurrency: 'EUR',       // codice ISO della valuta
    orderRevenue: 279.71,       // valore pagato dall'utente
    orderValue: 185.00,         // valore dei prodotti acquistati
    shippingAmount: 85.76,      // spese di spedizione
    status: 'COMPLETED,         // stato dell'ordine
    taxAmount: 13.95,           // tasse
    taxShippingAmount: 0.00,    // tasse su spedizione
    transactionId: 'a1b2c3'     // identificativo della transazione dell'ordine
}
```

Gli unici 2 campi del JSON obbligatori sono "**transactionId**" e "**status**".

Il Campo status può assumere i seguenti valori:

- **"NEW" -> avvio transazione: il primo evento UPDATE_CART crea implicitamente una transazione in questo stato, per cui questo evento è opzionale**
- **"PENDING" -> avvio checkout**
- **"COMPLETED" -> ordine completato con successo**
- **"CANCELED" -> ordine cancellato dall'utente**
- **"FAILED" -> ordine fallito (es. autorizzazione negata, errore nel pagamento, etc.)**
- **"EXPIRED" -> transazione scaduta (es. scadenza ordini tramite sistemi asincroni)**
- **"OFFLINE_PAYMENT" -> ordine completato con successo offline (es. pagamento in contrassegno)**

Si raccomanda infine che sia sempre valida la seguente formula:

---

*orderRevenue = orderValue + shippingAmount + taxAmount + taxShippingAmount + discountAmount (**attenzione, inviare sempre come numero negativo**)*

IMPRESSION:

Questo metodo è utilizzato per registrare il numero di volte che un prodotto viene visualizzato in una lista, banner o component all'interno del sito: incrociato con altre metriche, consente di ottenere informazioni importanti circa il proprio catalogo, le performance dei vari componenti (es. CTR) e il comportamento dei propri utenti all'interno del sito.

Il metodo "trackEvent" dovrà essere chiamato con il parametro "IMPRESSION" come segue:

---

pdxTrk.trackEvent('IMPRESSION', eventData);

Dove

---

var eventData = {

items: [

```
            {
                itemId: 123,                    // identificativo del primo prodotto
                originItemId: 'PRODUCT_LIST',   // componente che genera l'evento
            },
            {
                itemId: 456,                    // identificativo del secondo prodotto
                originItemId: 'PRODUCT_LIST',   // componente che genera l'evento
            },
            {
                itemId: 456,                    // identificativo del secondo prodotto in
altro componente della stessa pagina
                originItemId: 'TOP_BANNER,      // componente che genera l'evento
            }
        ]
}
```

È consigliabile legare questi eventi a strumenti che consentano di intercettare l'effettiva presenza e visualizzazione di un prodotto all'interno della pagina da parte dell'utente: un esempio di facile utilizzo è offerto dalla libreria JavaScript Waypoints.

Recognize User

Questo metodo è utilizzato per registrare le informazioni dell'utente rese disponibili in fase di login o registrazione sull'ecommerce.

In questo modo si ha la possibilità di associare i comportamente dell'utente ad uno o più indirizzi email.

Il metodo "recognizeUser" dovrà essere chiamato passando le informazioni dell'utente come segue:

```
pdxTrk.recognizeUser(userData);
```

Dove

```
var userData = {
```

```
    email: "mariorossi@mail.com",
    name: "Mario",
    surname: "Rossi"
}
```

Raccomandazione

SDK rende disponibili alcuni metodi che permettono di accedere facilmente alle funzionalità di raccomandazione di contenuti precedentemente configurati graficamente e associati alle campagne nella **"Dashboard Predixit".**

CALLBACK:

Il primo metodo permette di ricevere, su un metodo di callback specificato, le componenti grafiche della raccomandazione scelta inserendo l'id della campagna di raccomandazione come primo parametro e la funzione di callback da voi definita come secondo parametro.

Il metodo "**getRecommendation**" dovrà essere chiamato dunque come segue:

```
pdxTrk.getRecommendation('AB_1234567890123456789',
    myRecommendationCallbackFunction);
```

Dove la funzione di callback "myRecommendationCallbackFunction" deve accettare un unico parametro.

L'interfaccia del metodo in JavaScript nativo sarebbe:

```
function myRecommendationCallbackFunction(myObj)
```

Il parametro della funzione di callback sopra descritta sarà un JSON contenente il CSS e l'HTML del componente di raccomandazione.

```
myObj = {
        css: "…",
        script: "…",
        html: "…"
}
```

INSERIMENTO IN DOM:

Il secondo metodo disponibile permette l'inserimento automatico del componente di raccomandazione nella pagina, specificando l'id dell'elemento HTML che dovrà contenere la raccomandazione.

Il primo parametro atteso, come nel caso precedente, è rappresentato dall'id della campagna di raccomandazione, mentre il secondo parametro consiste in una stringa rappresentante l'id dell'elemento HTML che fungerà da contenitore della raccomandazione stessa.

Il metodo dovrà dunque essere chiamato come segue.

```
pdxTrk.printRecommendation('AB_1234567890123456789',
"myRecommendationElementId");
```

Automaticamente SDK inserirà le proprietà CSS come ultima inclusione dell'head della pagina e popolerà il contenitore HTML specificato con l'HTML erogato.

In questo caso si dovrà inserire nella propria pagina l'elemento HTML sopra citato, facendo attenzione che non esistano altri elementi con lo stesso ID.

Di seguito si fornisce un esempio dell'elemento HTML necessario.

```
<div id="myRecommendationElementId"></div>
```

Promozione

SDK rende disponibile facilmente l'accesso alle funzionalità di promozione precedentemente configurate graficamente e associate alle campagne nella **"Dashboard Predixit".**

CALLBACK:

Il metodo permette di ricevere su una callback specifica le componenti grafiche della promozione scelta inserendo l'id della campagna di promozione come primo parametro, un JSON contenente le opzioni di visualizzazione come secondo parametro e la funzione di callback da voi definita come terzo ed ultimo parametro. Si precisa che il secondo parametro può essere impostato a "null" nel caso non si vogliano inserire opzioni di visualizzazione

Il metodo "**getPromotion**" dovrà essere chiamato dunque come segue:

```
pdxTrk.getPromotion (
    'CD_1234567890123456789', //id promotion
    {modalPosition: 'center'}, //opzione di posizionamento del modal
    myPromotionCallbackFunction  //metodo di callback
```

```
);
```

Dove la funzione di callback "myPromotionCallbackFunction" deve accettare due parametri.

L'interfaccia del metodo in JavaScript nativo sarebbe:

```
function myPromotionCallbackFunction (myObjTemplate, myMetedata)
```

Il primo parametro della funzione di callback sopra descritta sarà un JSON contenente il CSS e l'HTML della raccomandazione.

```
myObjTemplate = {
        css: "",
        script: "",
        html: ""
}
```

Il secondo parametro invece, sarà un JSON contenente le informazioni configurate durante la procedura di creazione/modifica della promozione nella Dashboard.

```
myMetedata = {
  "promoModalPosition": "",  //posizione di visualizzazione del modale
  "promoTargetUrl": "" // url di redirect della promozione
  "promoWaitToView":  0 // attesa in millisecondi per la visualizzazione
  "promoScrollToView": 0 // altezza in px dello scroll necessaria per la visualizzazione
  "rendering.type": "modal" //tipologia di visualizzazione della promozione
}
```

Di seguito è fornita una breve descrizione dei campi sopra citati:

- **PromoModalPosition**, può assumere 9 valori differenti in base alla posizione scelta sullo schermo:
  - "tl" => top-left;
  - "tc" => top-center;
  - "tr" => top-right;
  - "ml" => middle-left;
  - "mc" => middle – center;
  - "mr" => middle – right;
  - "bl" => bottom-left;
  - "bc" => bottom – center;

  - "br" => bottom – right;

- **promoTargetUrl**, rappresenta l'url a cui si deve fare redirect all'atto del click sulla promozione;
- **promoWaitToView**, rappresenta in valore numerico i millisecondi che devono intercorrere dal caricamento della promotion nel DOM della pagina sino alla sua effettiva visualizzazione;
- **promoScrollToView**, rappresenta in valore numerico i pixel che devono scorrersi in altezza tramite scroll della pagina prima che la promotion sia effettivamente visibile;
- **rendering.type,** rappresenta la tipologia di visualizzazione della promozione, al momento della stesura di questo documento è settato su "modal";

DEFAULT CALLBACK:

Non specificando il metodo di callback sopra descritto, SDK utilizzerà il suo meccanismo di visualizzazione predefinito. Esso, tramite le funzionalità rese disponibili dal JavaScript nativo, aggiunge le proprietà CSS come ultima parte del tag HTML <head> della pagina e l'HTML del template come ultimo elemento del tag HTML <body>.

Usufruendo sempre delle sole funzionalità JavaScript native proporrà una visualizzazione del modale sul sito, tenendo conto di tutte le proprietà soprà descritte.

Si consiglia dunque l'utilizzo della funzionalità di callback specifica solo nel caso in cui si voglia proporre la promozione con tecnologia o grafica differente da quella di default.