Klarna

Network

Integration Guidelines

Version 1: 05/07/2024

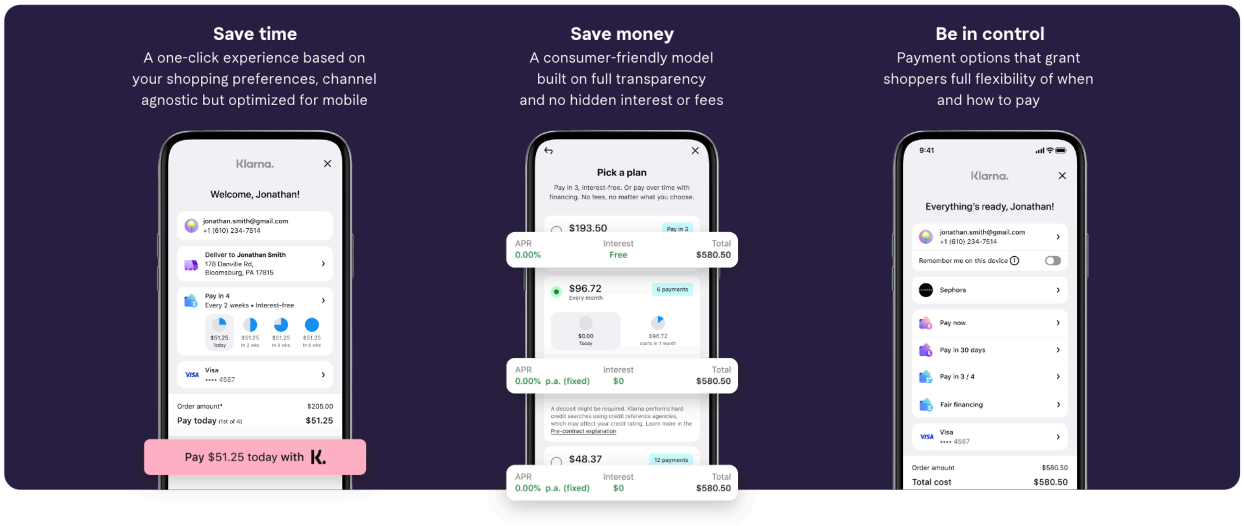
Table of Contents

Confidentiality disclaimer

Introduction

Understanding Klarna

Klarna is a global leading AI-powered payments network and financial assistant that smooths commerce by offering fairer, more sustainable, innovative solutions. We're committed to providing a seamless and secure shopping experience that help our shoppers:



Save time
A one-click experience based on your shopping preferences, channel agnostic but optimized for mobile

Save money
A consumer-friendly model built on full transparency and no hidden interest or fees

Be in control
Payment options that grant shoppers full flexibility of when and how to pay

Power your growth with Klarna

We've partnered with global payment platforms to make Klarna's flexible and convenient payment option the default checkout choice for shoppers worldwide. Each month, millions of shoppers opt for Klarna for their transactions, both online and in physical stores.

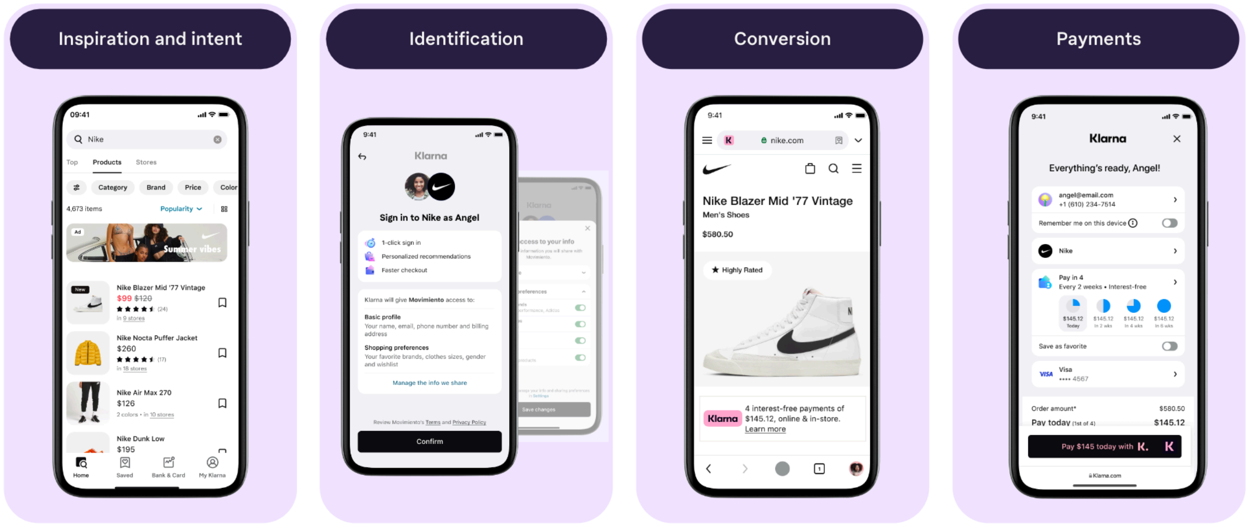
150M Shoppers	**2.5M** Daily transactions	**+550K** Merchants globally

Merchants grow their business with our flexible payment option and smart shopping solutions that enable shoppers to easily and securely pay when and how they want everywhere - online and in-store, powered by the Klarna App. Klarna supports all shopping scenarios from high value transactions to everyday purchases. Merchants using Klarna see:

- **41%** Increase in average transaction value.
- **30%** Increase in conversion.
- **45%** Higher purchase frequency than average shoppers.

Klarna elevates the shopping journey from inspiration and intent to checkout and retention. We prioritize streamlining payments for merchants with a simplified checkout process, conversion optimization, and shoppers identification.

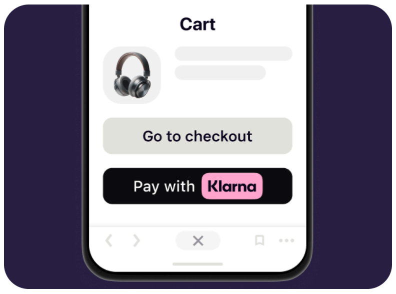
Our dynamic expertise ensures seamless interoperability between products within the Klarna ecosystem, delivering a smooth and consistent shopper experience across all integration patterns.

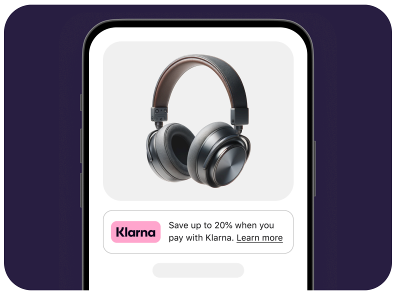

Inspiration and intent
Identification
Conversion
Payments
Sign in to Nike as Angel
Nike Blazer Mid '77 Vintage
4 interest-free payments of $145.12, online & in-store.
Everything's ready, Angel!
Pay $145 today with Klarna

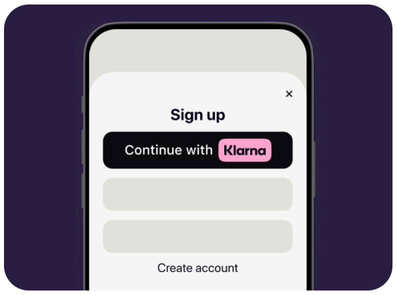
The Klarna Product ecosystem: We elevate the shopping journey from start to end.

Smart solutions to maximize sales

Our offering includes On-site messaging, Klarna Express checkout, and Sign in with Klarna alongside marketing services such as affiliation and price comparison search. These features are exclusively designed to drive growth and boost conversion rates and enhance the shopper experience. Partnering with us ensures seamless interoperability, mirroring direct integration with Klarna for a cohesive experience.







Klarna Express checkout

Offer a **6x faster** check-out process that will lower the threshold for shoppers to complete a purchase.

On-site messaging

Add personalized messaging throughout the shopper journey for **higher conversion rates and increased spend**.

Sign in with Klarna

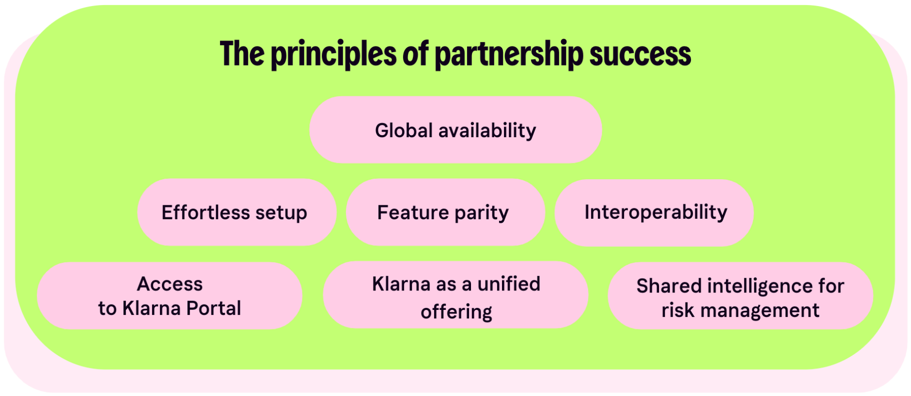
Accelerate the registration, sign-in, and checkout processes with a **one-click experience**, increasing account registrations, **and improving conversion rates.**

Before you start

Klarna offers a global and interoperable platform designed to support merchant management, transaction processing, post-purchase operations, and conversion rate optimization. This platform is powered by the Web SDK, an extensive JavaScript SDK, along with the Management and Partner Product APIs. These tools ensure seamless integration, enabling you as a Klarna partner to maximize the benefits of all available Klarna features.

Klarna aims to partner with you as an Acquiring Partner integrating partner and unlock our mutual potential to enable best in class shopping experiences that allow merchants to maximize results as well as offer a unique customer journey for shoppers.

As an Acquiring Partner integrating partner, understanding our solution principles and committing to them is essential before proceeding with integration and offering Klarna services to your merchants. This understanding ensures that you can provide a best-in-class experience to your merchants, streamlining the integration and the adoption of future enhancements.


The principles of partnership success
Global availability
Effortless setup
Feature parity
Interoperability
Access to Klarna Portal
Klarna as a unified offering
Shared intelligence for risk management

- **Global availability**
    - Klarna solutions are global by design ensuring that partners can seamlessly integrate and enable all Klarna's services on a worldwide scale right from the start. As Klarna is on a rapid expansion path, with plans to reach 200 markets in the future, it's crucial for partners to adopt a global perspective in both technical and commercial aspects of integration. This approach eliminates the need for additional adjustments when Klarna enters new markets or expands services in existing ones. Embracing a global framework from the outset not only simplifies integration but also positions you as a Klarna partner to effortlessly grow alongside with our continuous expansion.

- **Effortless setup**
    - Klarna solutions are available to any merchant and can be seamlessly integrated into any system where traditional card payments are accepted, without the need for extra data. Klarna offers all the functionalities of card payments and additional flexibility for shoppers while providing a streamlined payment experience. By adding Klarna services into your offering, you're not merely adding another way to pay; you're

enabling merchants to enrich the shopping journey for shoppers and keeping it simple and efficient.

- **Feature parity**
  - Merchants should be able to effortlessly activate all Klarna services without facing extra technical complexities or functional limitations, no matter the integration pattern. By aligning your integration with Klarna's best practices and supporting Klarna's Web SDK, mobile SDK, as well as enablement of Klarna services for all channels offered via your platform and payments use cases offered such as in-store payments, subscriptions will enable you to easily expand your business and offer additional value to your merchants.

- **Interoperability**
  - Klarna solutions strive to create intuitive and consistent shopper experiences independently of how they are integrated (e.g. API, platforms, etc.). Klarna's products will be enabled to connect seamlessly with each other and with partner platforms, ensuring all merchants, regardless of their technical setup, can fully use these integrations to enrich shopper experiences by utilizing Klarna's full product offering

- **Access to Klarna Portal**
  - A key part of ensuring feature parity and interoperability is having your merchants access the Klarna portal, through which merchants can enable and configure growth and marketing services. The portal can also serve features like Klarna dispute handling, if you choose not to integrate such into your own portal/dashboard.
- **Klarna as a unified payment offering**
  - Merchants should offer Klarna as a single payment method, giving shoppers access to various payment options. The goal is to provide shoppers with maximum freedom and flexibility in how they want to pay using Klarna. The specific payment options from Klarna are curated based on local rules and regulations and are determined by the shopper at the time of purchase.

- **Shared intelligence for risk management**
  - Our common goal is to protect merchants and shoppers from fraud by leveraging shared intelligence. Through collaboration with our partners, we exchange data and insights to improve the accuracy of our risk analysis and enhance our collective efforts in risk management.

APIs overview

Management API

Global and instant onboarding, full lifecycle management of accounts

- Create, read, update and onboard merchant accounts.
- Fetch and verify price plans at any time.
- Integrated, automated fraud management.
- Flexible settlements to simplify reconciliation process
- Restricted to distribution or acquiring partners.

Partner product API

Unified global service to simplify enablement of Klarna offering

- Single, global authentication for all features.
- One API to access end-to-end services and features to process Klarna payments.
- Standardized data point definitions.
- Available for distribution partners and merchants, interoperable.

Integration overview

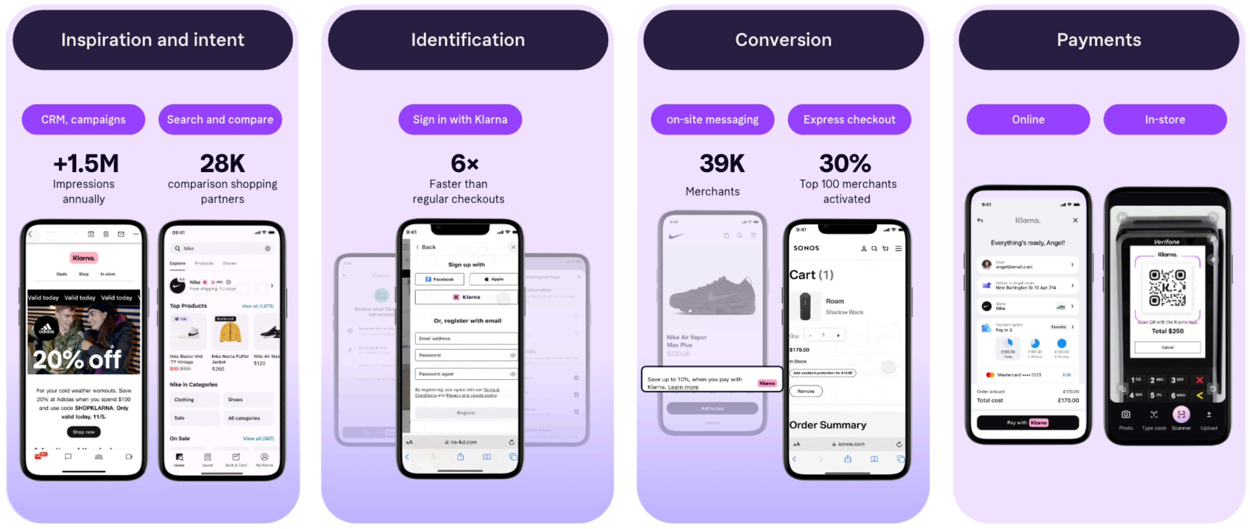
These guidelines are divided into seven different sections covering the complete integration journey. Below you will find quick links to each of these sections.

1. Designing your Klarna Solution
    a. Building Shopper-centric solution
    b. Presenting Klarna
2. Get your partner account setup and ready to start your integration:
    a. Obtain API Credentials: Get your API credentials from Klarna.
    b. Configure Webhooks: Set up webhooks for real-time updates.
    c. Manage Account Configuration: Adjust your account settings.
3. Discover Management API.
    a. Create Merchant Accounts: Set up your first merchant account.
    b. Update and Maintain Merchant Accounts: Integrate additional requests for account management.
    c. Enable Access to Klarna Merchant Portal: Optimize conversion rates for your merchants.
4. Enabling interoperability of Klarna products
    a. Importance: Understand why interoperability is crucial.
        i. Klarna Shopper Journey: Learn how to enhance the shopper experience.
    b. How to Do It: Steps to achieve interoperability.
        i. Interoperability: Ensure seamless integration of Klarna products.
        ii. Klarna Portal (Deep-links): Utilize deep-links for efficient navigation.
5. Discover Partner Product API: Explore the Partner Product API.
    a. Designing Your Klarna Solution
    b. Follow Best Practices: Ensure the UX is optimized for conversion.
    c. Identify Integration Approach: Choose the ideal integration method.
        i. Create a Payment Request: Initiate a payment request.
        ii. Monitor Payment Status: Keep track of payment requests.
        iii. Confirm Payment: Verify the payment request.
    d. Explore Payment Scenarios: Address specific payment requirements.
    e. Manage Authorizations: Implement requests for captures, voids, refunds, and updates.
6. Enable Post Purchase Operations: Facilitate seamless post-purchase activities.
    a. Handle Disputes: Implement disputes handling.
    b. Create Settlement Reports: Allow for settlements reporting.
7. Finalize your Klarna Integration
    a. Validate Your Integration: Ensure your integration meets all requirements.
    b. Create Public Documentation: Provide documentation for your merchants.
8. Klarna API Design Principles: Adhere to Klarna's API design standards.

Design your Klarna Solution

Designing your Klarna solution extends beyond just adding components. It's about forming a partnership to elevate the overall shopper experience, grow shopper satisfaction and improve the performance of every step of the purchase journey.

In the following sections you will find details on the main areas of focus to keep in mind when designing your solution for merchants.



The Klarna Product ecosystem: One dynamic experience, seamless Interoperability between products across integration patterns.

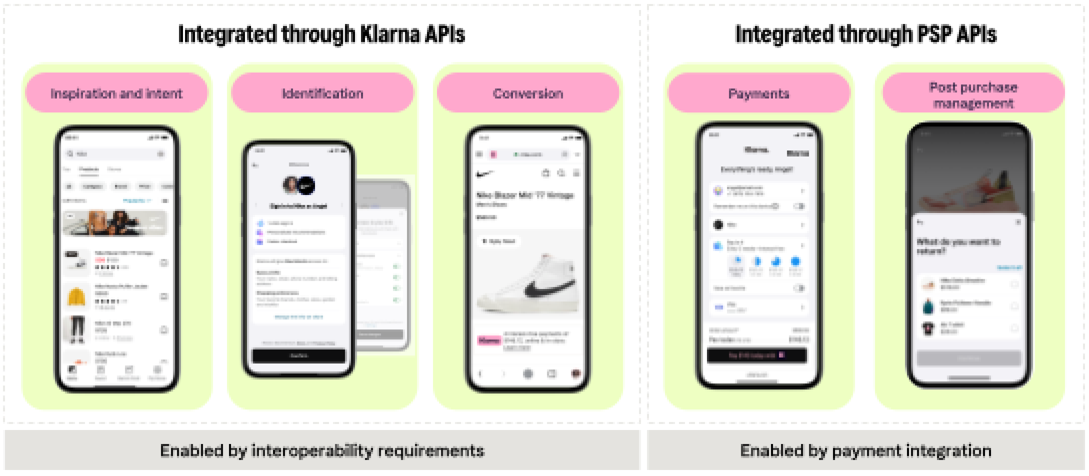
Klarna Product Interoperability

In today's fiercely competitive market, it's crucial that every shopper who lands on your merchant's website makes a purchase to ensure continued success. To help merchants achieve this, we've developed a product framework that guarantees seamless interoperability, allowing all systems involved in merchant integrations to work together effortlessly. Not only does this streamline the merchant's operations, but it also enhances the overall shopping journey, ensuring a best-in-class shopping experience, and global consistency.

Merchants have full access to the complete suite of Klarna products and services within this framework. By enabling the recommended integration flow outlined later in this document, merchants can:

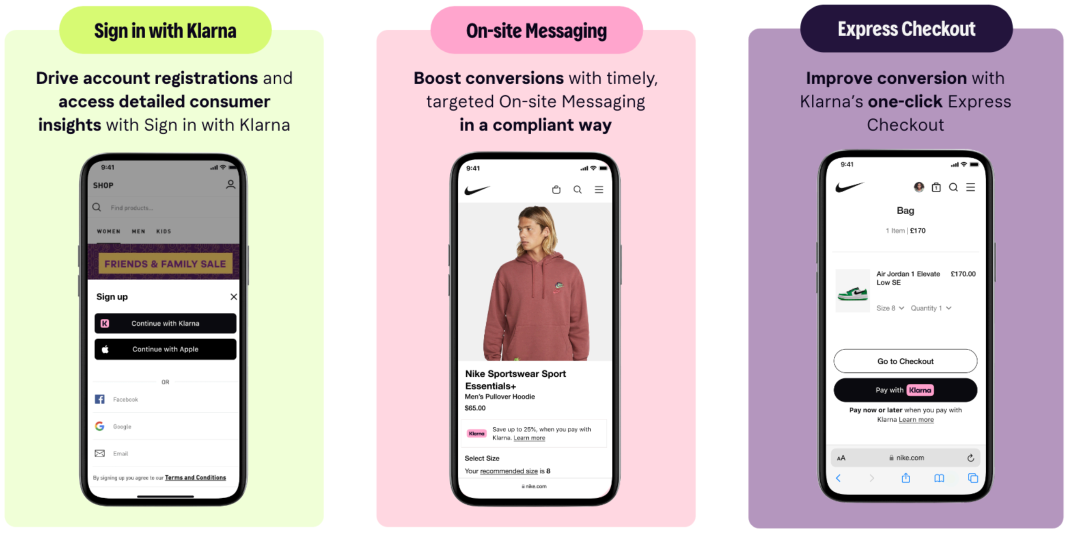
- **Flexibility**: Tailor the complete range of Klarna services to their unique needs.
- **Better Conversion Rates**: Leverage conversion-boosting Klarna features to reduce shopper drop-offs and increase the likelihood of successful purchases, regardless of the integration approach for processing payments.

- **Higher Shopper Satisfaction**: Provide a reliable and familiar payment experience that enhances trust and satisfaction, encouraging shopper retention.


Integrated through Klarna APIs
Integrated through PSP APIs
Inspiration and intent
Identification
Conversion
Payments
Post purchase management
Enabled by interoperability requirements
Enabled by payment integration

Solutions accessible via interoperability

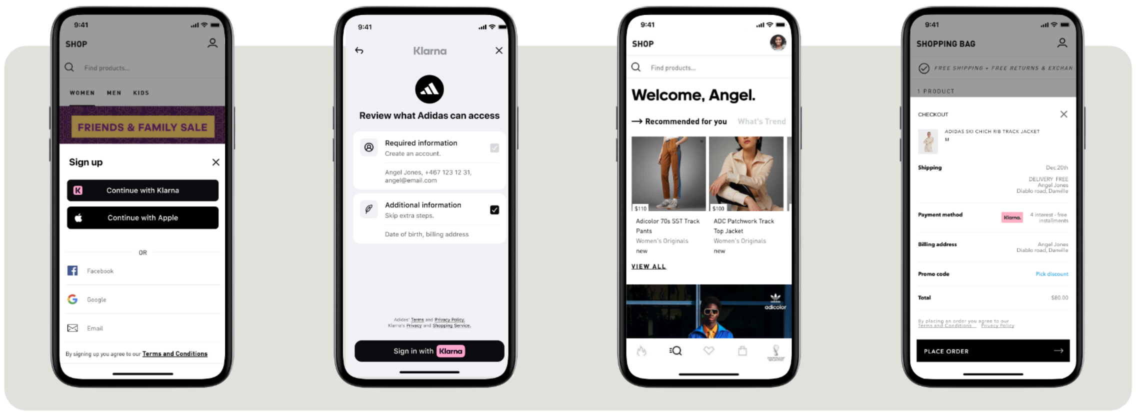
Sign in with Klarna, On-site messaging and **Express Checkout** are Klarna's conversion booster solutions. These were designed to enhance the shopper experience at different touchpoints, before and during the purchase. Each feature targets a specific aspect of the shopping journey, and they deliver maximum impact when utilized together.


Sign in with Klarna
Drive account registrations and **access detailed consumer insights** with Sign in with Klarna
SHOP
FRIENDS & FAMILY SALE
Sign up
Continue with Klarna
Continue with Apple
On-site Messaging
Boost conversions with timely, targeted On-site Messaging **in a compliant way**
Nike Sportswear Sport Essentials+
Men's Pullover Hoodie
$65.00
Select Size
Express Checkout
Improve conversion with Klarna's **one-click** Express Checkout
Bag
Air Jordan 1 Elevate Low SE
Go to Checkout
Pay with Klarna

Tools to help merchants achieve maximum growth.

Sign in with Klarna

Merchants are constantly aiming to enhance the purchase experience for their shoppers. By leveraging Klarna's community of over 150M shoppers, this social login feature lets shoppers quickly and safely sign up on the merchant website by using their Klarna account information and allows merchants to identify their shoppers early in the shopping journey.



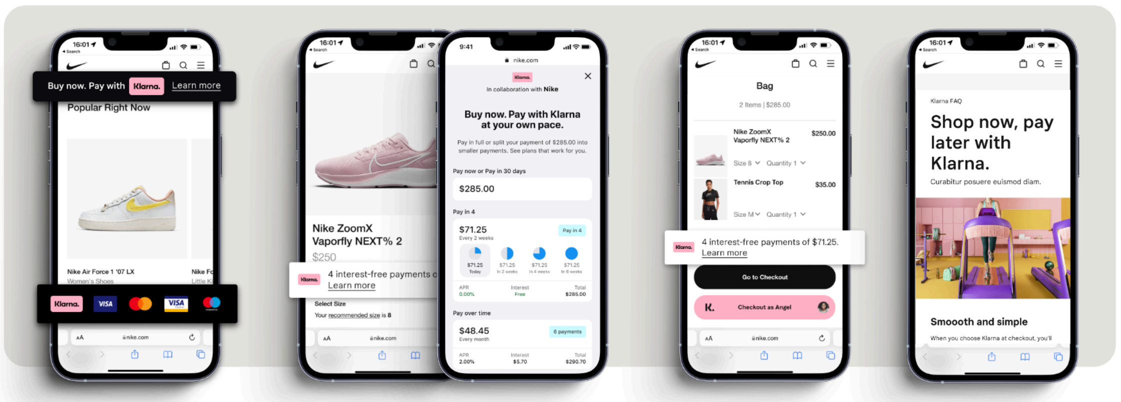
Sign-in page Klarna review page Merchant page Checkout page

Easy registration, sign-in and checkout. Plus unrivaled access to detailed shopper data.

When a shopper registers with Klarna in an ecommerce site, the merchant gains additional intelligence that will enable the enhancement of the shopping journey, understanding the shopper's needs and delivering a personalized experience.

On-site messaging

Klarna's dynamic placement solution, On-site messaging, helps your merchants business grow by converting website visitors into shoppers by informing early in the shopping journey about flexible payment methods available.


Buy now. Pay with Klarna. Learn more
Popular Right Now
Nike ZoomX Vaporfly NEXT% 2
4 interest-free payments
Buy now. Pay with Klarna at your own pace.
Bag
4 interest-free payments of $71.25. Learn more
Go to Checkout
Checkout as Angel
Shop now, pay later with Klarna.
Smoooth and simple

Sitewide Product page Cart FAQ

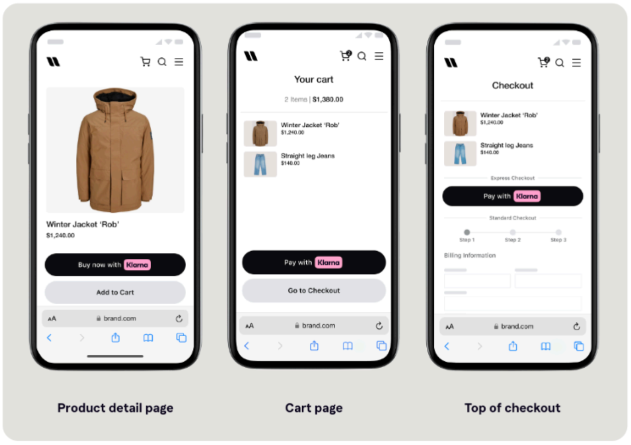
On-site messaging touchpoints: By adding On-site messaging to the shopping journey, merchants can inform shoppers about promotions, available payment methods and a payment calculator.

The look and feel for these dynamic placements is customizable and merchants are in control to choose their preferred font, text style, size and logos allowing them to match the look and feel of their overall brand and website.

Express checkout

Klarna Express checkout allows your merchants to uplift conversion and minimize cart abandonment by pre-filling the shopper's at the checkout moment and providing a faster and more enjoyable shopping experience.

Merchants are able to place Klarna Express checkout where shoppers are most likely to complete a purchase and display it early in the shopping journey to provide the option to skip ahead when they're ready to purchase.



More Information on Interoperability is available in Enable interoperability of Klarna products.

How to present Klarna in the checkout

Checking Klarna availability

To enhance global scalability, Klarna enables Partners to verify the suitability of Klarna payments during checkout before displaying our branding. This approach ensures that as Klarna expands into new markets, you only need to update your merchant configurations within Klarna, simplifying the process of scaling up and ensuring a truly global partnership. This setup also facilitates a seamless merchant experience globally.

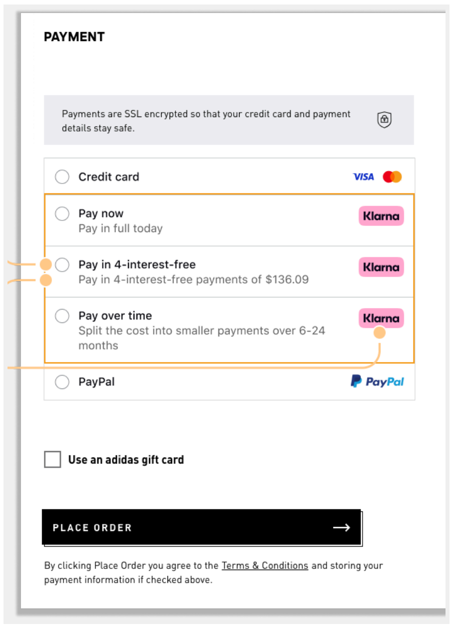
The core principle of integrating Klarna in the checkout process is the dynamic display of content, which adjusts based on this initial verification for scalable results. To support this dynamic capability, the Klarna Messaging API package is designed to let you dynamically display accurate payment descriptors based on your account settings and transaction specifics. This tool provides crucial information and visuals, helping you effectively showcase Klana-branded elements to enhance shopper conversion rates.

Checkout structure

Get familiar with the different components of the checkout page and how to display Klarna in the merchant payment selector. There are three main items to consider:

- Payment descriptor
- Payment subheader
- Klarna badge

These may vary depending on the language and market. Klarna will provide these dynamically as part of the API response.



PAYMENT

Payments are SSL encrypted so that your credit card and payment details stay safe.

Credit card

Pay now
Pay in full today

Pay in 4-interest-free
Pay in 4-interest-free payments of $136.09

Pay over time
Split the cost into smaller payments over 6-24 months

PayPal

Use an adidas gift card

PLACE ORDER

By clicking Place Order you agree to the Terms & Conditions and storing your payment information if checked above.

Two options are available for presenting Klarna in checkout, depending on your capability to dynamically handle the presentation of Klarna in checkout

Option 1: Dynamic (recommended)

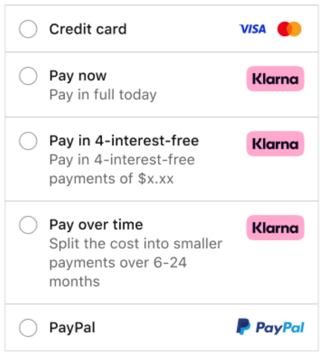
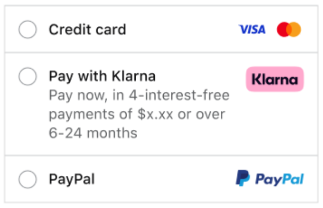
For integrations capable of dynamically providing payment methods based on Klarna's response, the recommended approach that allows optimization of conversion is to present each payment option available as specified in the response.

This approach ensures clarity for shoppers regarding available options and enables further flexibility to adapt to additional countries as they are enabled, supporting a future-proof and resilient integration. In addition, this allows for the presentation of deals to the end customer, allowing Klarna to drive increased conversion in your merchant checkout.

Option 2: Static

If your integration or that of your merchants cannot dynamically display Klarna payment options, present Klarna as a single payment choice labeled "Pay with Klarna." This method guarantees the correct representation of Klarna across all markets and to all shoppers, maintaining consistency and simplicity.

These methods provide merchants with the flexibility needed for a globally adaptable checkout experience, aligning with their various shopper experience requirements and expectations.

Dynamic Option (Recommended)	Static Option
	
Credit card — VISA Pay now — Pay in full today — Klarna Pay in 4-interest-free — Pay in 4-interest-free payments of $x.xx — Klarna Pay over time — Split the cost into smaller payments over 6-24 months — Klarna PayPal — PayPal	Credit card — VISA Pay with Klarna — Pay now, in 4-interest-free payments of $x.xx or over 6-24 months — Klarna PayPal — PayPal

Pay with Klarna button

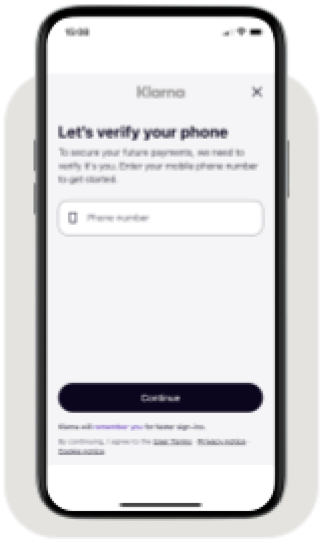
Allowing the shopper to complete their purchase by using Klarna's JavaScript SDK simplifies the checkout process. By implementing the Klarna payments package as detailed here, you can easily display the payment button and handle the necessary actions when it's clicked. At this point the payment process is taken over by Klarna, enabling shoppers to proceed with their transactions efficiently.

Payment flow

Once the shopper confirms the intention to pay with Klarna by clicking on the button, they will be redirected to Klarna payment flow. The initiation of the flow will vary depending on the integration approach:

- Klarna.js integration
- Server-side only integration

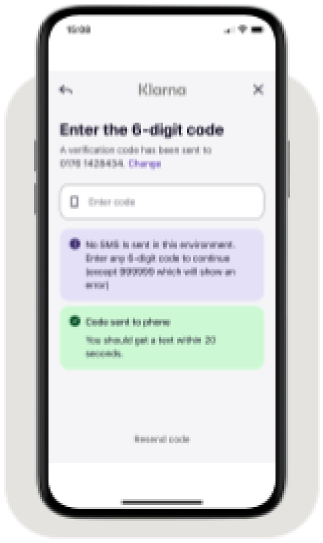
Klarna payment flow provides shoppers with a smart, consistent, and predictable experience, regardless of where they shop or how they want to pay. It offers a seamless process for both new and returning shoppers, as account creation is handled within Klarna's modal.



Klarna

Let's verify your phone

To secure your future payments, we need to verify it's you. Enter your mobile phone number to get started.



Klarna

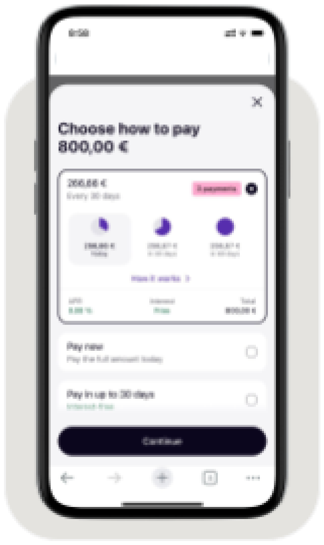
Enter the 6-digit code

A verification code has been sent to 0176 1428434. Change

No SMS is sent in this environment. Enter any 6-digit code to continue (except 999999 which will show an error)

Code sent to phone

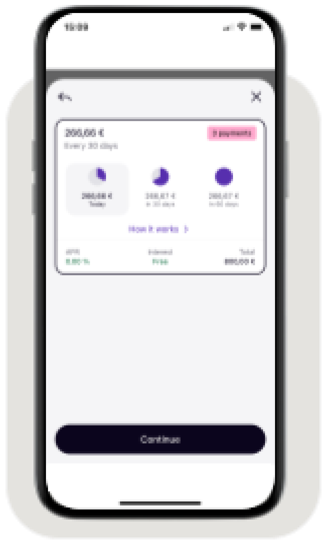
You should get a text within 20 seconds.



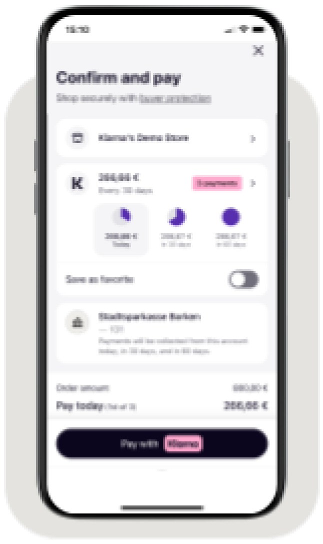
Choose how to pay
800,00 €

After verifying their phone number, New Klarna shoppers would need to provide their email, billing address and additional personal data.
Data points can be prefilled if supplied by the integrator when initiating the payment request.

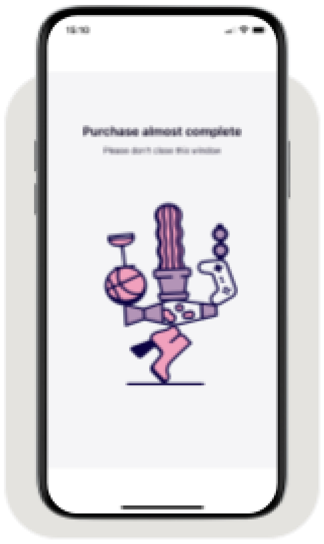
Upon successful authentication, the shopper will be invited to select one of the payment options offered by Klarna.



A payment plan will be shown to the shopper to review (this plan will vary based on the selected payment option)



The review screen will allow the shopper to check all the setup definitions and complete the payment.



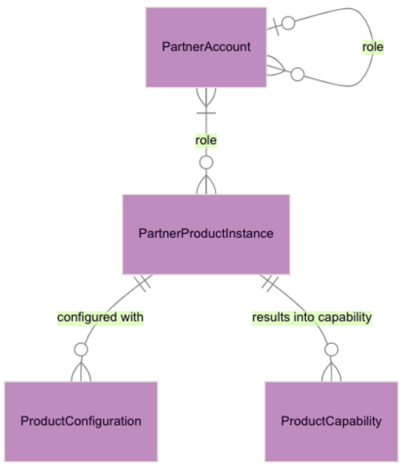
Finally a confirmation screen will be displayed before redirecting the shopper to the `redirect_url` provided in the payment request.

Set up your partner account

Partner account

A partner account is a construct that consolidates all information, capabilities, and features based on agreements made with a partner of Klarna. It acts as a central repository for various types of data related to a partner, similar to a folder on a computer that can store different types of files.

A partner account can reference multiple partner product instances, each configured with specific capabilities and configurations. For instance, a payments product instance might have roles for a merchant and an acquiring partner, with each role having specific responsibilities.



Step 1: Configure account credentials

To begin your integration with Klarna, the first step is to obtain your API credentials. Once your account is set up by Klarna, we will provide you the first API key through a secure link.

When you have your initial API key from Klarna, you will be able to create new API keys and Client IDs through the Management API.

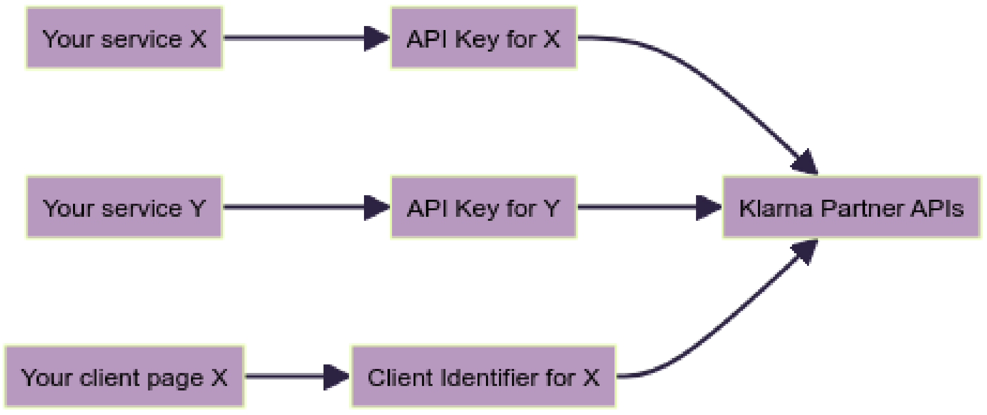
- **API keys:** Are used to authenticate server-side REST API requests towards Klarna. In addition, Klarna may use them to identify the source account.
  - Structure: `klarna_<live|test>_<api>_<random>`
- **Client IDs:** Are used to authenticate client-side interactions towards Klarna's SDK.
  - Structure: `klarna_<live|test>_<client>_<random>`
  - Due to the nature of frontend authentication, client keys require domain registration.

To learn more about authentication, API keys, Client IDs and Security consult the **Authentication** Section.

Account credential management

Credential management is entirely under your control, allowing you to create and manage these for different services. This setup enhances security by enabling the regular rotation of credentials automatically, eliminating the need for manual intervention by Klarna.

To minimize risk in the event of a security breach, it's advisable to **assign distinct credentials to each of your services**. If one credential is compromised or needs to be disabled, it will not affect the others, ensuring continuous operation across your integration.



Credentials can be created and managed for live or test mode, and are specific to either client-side or server-side actions. When creating credentials, you can include a description to clarify their intended use case. This description can later be verified through a GET request to `/v1/account/credentials`.

For rotating credentials, it's recommended to support multiple credentials during the transition. The steps for key rotation involve:

- Creating new credentials.
- Updating the existing credentials to the new credentials.

- Validate the new credentials have been correctly implemented before making a DELETE request to `/v1/account/credentials/{credential_id}` to permanently disable the affected credential.

Consult the **API reference** for a complete description of the request body parameters.

Rate limiting considerations:
Rate limiting is enforced by Klarna on an account basis. The creation of multiple credentials will not enable increased rate limits. For more information see Rate Limiting.

⚠️ If credentials are not used for two months, they will be disabled to prevent misuse, and will be deleted after ten months of inactivity. In such cases, credentials can be reactivated or new keys created via Partner Support, your PST or through APIs, maintaining the security and flexibility of your Klarna interactions.

Step 2: Configure Klarna webhooks

As part of a Klarna integration webhooks allow your applications to receive business event notifications as they occur in integrated Klarna services, so that your backend systems can react in real time.

The webhooks are customizable, and all of the Klarna APIs come with a standard set of webhook events that can be subscribed to.

Refer to the "Events categories" section for supported event types.

To start, set up and configure webhook events, and target URLs, see the "Getting Started with Klarna Webhooks guide" guide.

Consult the **API reference** for a complete description of the request body parameters.

Getting started with Klarna webhooks

To ensure your systems are prepared to receive and handle Klarna notifications for uninterrupted integration with Klarna's services, follow the below process:

1. **Establish a secure endpoint**
   a. Expose an HTTPS endpoint on your server designed to receive webhook notifications.
   b. Only HTTPs endpoints are supported, and a valid SSL certificate is required.
2. **Endpoint Configuration**
   a. Create a single endpoint for multiple event types or designate separate endpoints for each event type according to your preference.
   b. See more info in the Create and manage webhooks section.
3. **Receive Notifications**
   a. Verify HMAC Signature to ensure data security and integrity
   b. See more info in the Create and manage signing keys section.

4. **Store the webhook event data for further processing.**
    a. You will receive Klarna notification to the configured webhook endpoint.
    b. Example of webhook payload

```JavaScript
{
"metadata": {
  "event_type": "payment.request.state-change.{event_type}",
  "event_id": "{{unique event UUID}}",
  "event_version": "v1",
  "occurred_at": "2024-01-01T12:00:00Z",
  "correlation_id": "{{unique correlation UUID}}",
  "account_id": "{{account-specific ID}}",
  "product_instance_id": "{{product-specific ID}}",
  "webhook_id": "{{webhook-specific ID}}",
  "live": {{boolean}}
},
"payload": {{content of the payload varies according to the event type. More
information available in product-specific subsections}}
}
```

5. **Acknowledge the webhook**
    a. To confirm successful delivery, responses to webhook notifications must be immediate and should carry an HTTP status code of 200, 201, 202, or 204.
    b. Failure to respond or a response with a different status code, will be treated as an unsuccessful operation by Klarna, triggering the retry mechanism of the notification at increasing intervals.
6. **Retry mechanism**
If Klarna receives a timeout or HTTP 4XX or 5XX response codes from your system, the notification delivery is deemed failed, and Klarna will initiate retries based on its webhook retry policy. Klarna orchestrates these retries with progressively longer delays, which can extend up to 12 hours or until a successful response is received.

Here's how the retry schedule is structured:

- The first retry occurs 10 seconds after the initial failure.
- If the first retry is unsuccessful, a second attempt follows 2 minutes later.
- Subsequent failures trigger another retry after 15 minutes.
- If the issue persists, retries are scheduled at 3 hours, 6 hours, and finally 12 hours for the last attempt.

This structured approach ensures multiple opportunities for notifications to succeed, enhancing the reliability of the communication between systems.

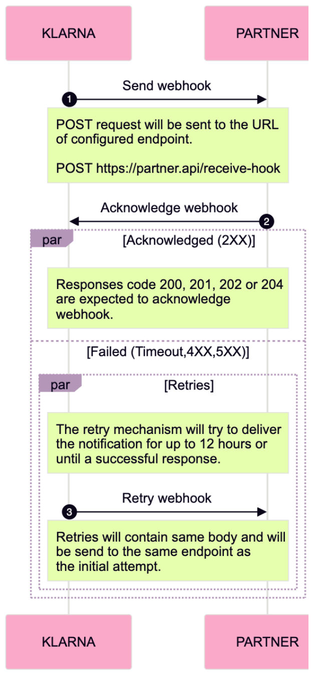
Handling failed notifications

- If no successful response is received after the final retry attempt at 12 hours, the notification will be considered permanently failed.

Manual processing of failed notifications

- In the event that a partner resolves an issue on their end and wishes to receive the previously failed notification, they should contact our support team. Our support team can manually process the failed notifications upon request.

When you modify webhook settings during ongoing retries, these changes will only apply to new notifications. If a triggered Klarna webhook fails to receive a response code of 200, 201, 202, or 204, it will continue to retry using the old configuration until it either successfully communicates or reaches the 12-hour retry limit.


KLARNA
PARTNER
Send webhook
1
POST request will be sent to the URL of configured endpoint.
POST https://partner.api/receive-hook
Acknowledge webhook
2
par
[Acknowledged (2XX)]
Responses code 200, 201, 202 or 204 are expected to acknowledge webhook.
[Failed (Timeout,4XX,5XX)]
par
[Retries]
The retry mechanism will try to deliver the notification for up to 12 hours or until a successful response.
Retry webhook
3
Retries will contain same body and will be send to the same endpoint as the initial attempt.
KLARNA
PARTNER

⚠️ To avoid disruptions and ensure a smooth transition, it is recommended to initiate and run a new webhook in parallel before discontinuing an older webhook. This strategy ensures that the new settings are fully operational and effective, maintaining seamless notification delivery during the transition period.

Create and manage signing keys

To securely receive webhook notifications, you must generate signing keys and attach them to your webhooks. This process ensures that you can verify the authenticity of incoming notifications. Klarna will use the signing key to sign the notification before sending. The signature and the identifier will be part of the notification, allowing you to validate the notification. When setting up a new webhook, indicate which signing key to use.

Signing keys considerations

Once Signing Keys are generated, remember to store the signing key as you won't be able to view it again for validating webhooks.

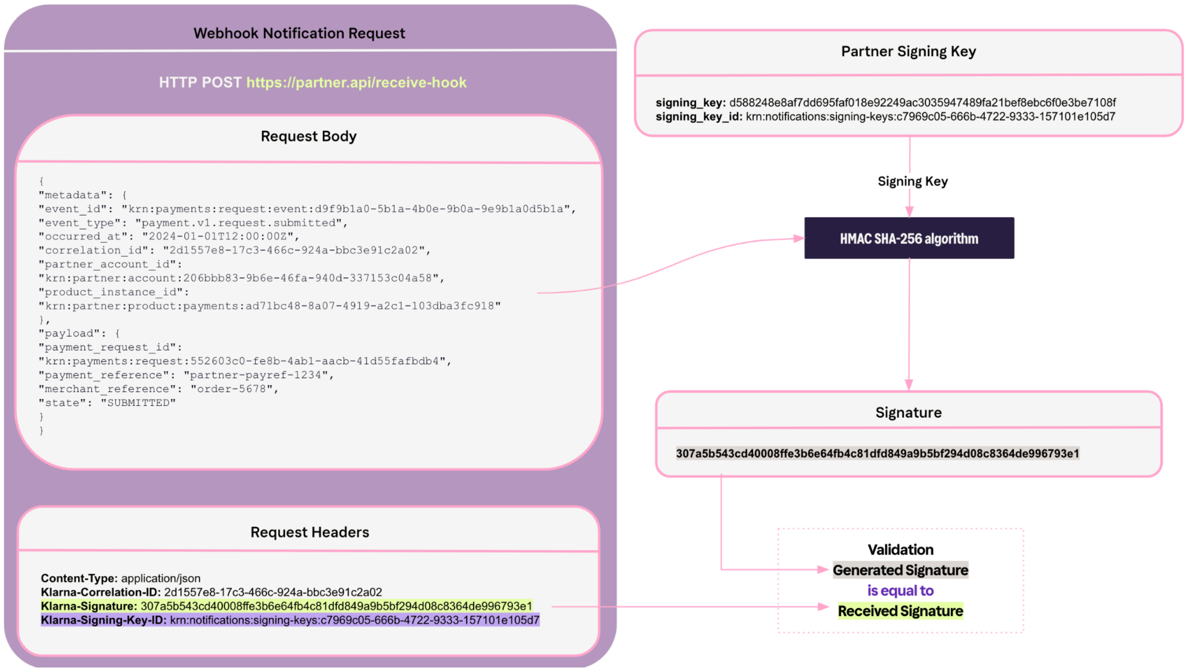
Verify HMAC signature to ensure data security and integrity

1. Compute your signature:
    a. Serialize the JSON in the HTTP request body into a string, removing all whitespaces and newlines.
    b. Use the `signing_key_id` from the header to identify the appropriate signing key.
    c. Apply the HMAC-SHA256 algorithm to the serialized string using the signing key.
2. Parse `Klarna-Signature`:
    a. Extract the "`Klarna-Signature`" from the HTTP headers of the received request.
3. Compare signatures:
    a. Match your computed signature against the "`Klarna-Signature`" from the HTTP headers.
4. Validate the request:
    a. Confirm the request's authenticity if the signatures align. If not, reject the request with an HTTP 400 Bad Request response to indicate potential tampering.

⚠️ *Currently there is a maximum of 50 signing keys per account. In case you need to create new keys, it is required to remove unused signing keys through a DELETE request to* `/v1/notification/signing-keys/{signing_key_id}`.

Response example:

```
JavaScript
{
"signing_key_id":
"krn:partner:global:notification:signing-key:00000000-0000-0000-0000-0000000000
00",
"signing_key": "00000000-0000-0000-0000-000000000000",
"created_at": "2024-01-01T12:00:00Z"
}
```



```
{
"metadata": {
"event_id": "krn:payments:request:event:d9f9b1a0-5b1a-4b0e-9b0a-9e9b1a0d5b1a",
"event_type": "payment.v1.request.submitted",
"occurred_at": "2024-01-01T12:00:00Z",
"correlation_id": "2d1557e8-17c3-466c-924a-bbc3e91c2a02",
"partner_account_id":
"krn:partner:account:206bbb83-9b6e-46fa-940d-337153c04a58",
"product_instance_id":
"krn:partner:product:payments:ad71bc48-8a07-4919-a2c1-103dba3fc918"
},
"payload": {
"payment_request_id":
"krn:payments:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
"payment_reference": "partner-payref-1234",
"merchant_reference": "order-5678",
"state": "SUBMITTED"
}
}
```

```
Content-Type: application/json
Klarna-Correlation-ID: 2d1557e8-17c3-466c-924a-bbc3e91c2a02
Klarna-Signature: 307a5b543cd40008ffe3b6e64fb4c81dfd849a9b5bf294d08c8364de996793e1
Klarna-Signing-Key-ID: krn:notifications:signing-keys:c7969c05-666b-4722-9333-157101e105d7
```

signing_key: d588248e8af7dd695faf018e92249ac3035947489fa21bef8ebc6f0e3be7108f
signing_key_id: krn:notifications:signing-keys:c7969c05-666b-4722-9333-157101e105d7

307a5b543cd40008ffe3b6e64fb4c81dfd849a9b5bf294d08c8364de996793e1

Consult the **API reference** for a complete description of the request body parameters.

To handle changes such as the rotation of signing keys, it's recommended to support multiple signing keys during the transition. The steps for key rotation involve:

- Creating a new signing key.
- Updating the recipient server to accept the new signing key.
- Updating the webhook.
- Validate the new signing key has been correctly implemented removing the old signing key.

Create and manage webhooks

Create webhooks to receive notifications about configured events. These notifications are sent to the provided endpoint and signed with the corresponding signing key.

You are able to use wildcards to set up the desired event types.

Webhook wildcards
By using a wildcard such as "*", all events that match the specified pattern will be included.
Example: with "`payment.request.*`", you will receive all events associated with payments like:
- `payment.request.state-change.submitted`
- `payment.request.state-change.in-progress`
- `payment.request.state-change.prepared`
- `payment.request.state-change.authorized`
- `payment.request.state-change.canceled`
- `payment.request.state-change.expired`
- `payment.request.updated`

Events categories

Here are the categories of events you can receive through webhooks:

- **Payment request**: Notifications of state changes on the lifecycle of a payment request.
- **Payment transaction**: Notification of state changes on the lifecycle of a payment transaction.
- **Payment dispute**: Notifications of state changes of the dispute lifecycle, for example, when a dispute is initiated, escalated, or resolved.
- **Accounts**: Notifications when a merchant's account changes status.
- **Settlements**: Updates about the settlement process, such as a payout has been done or a settlement report is ready for download.

Once created, you can use the API requests to manage webhooks and update, delete or list all details previously configured.

Consult the **API reference** for a complete description of the request body parameters.

Specific webhooks associated with individual components of your Klarna integration are listed within the integration guidelines for those products. Please find detailed information on the integration of those webhook components below:

- Account lifecycle and management webhooks
- Client-side payment request notification events
- Server-side payment request notification events
- Transactional Dispute webhooks
- Settlement webhooks

Simulate and test webhooks

In order to test your webhook integration, you can simulate a webhook by manually triggering an event that you have subscribed to. Provide the `webhook_id`, `event_type`, and `event_version` for the webhook previously configured to endpoints on your account.

This feature is available in the test environment. We recommend testing your webhook integration before going live. By manually triggering webhooks, you can instantly simulate any state change for a payment request without going through the purchase flow or waiting for the request to expire. This allows you to test creating, confirming, and changing the state of payment transactions. You can also receive notifications of expired payment requests for abandoned cart or similar retargeting purposes.

In the test environment, the simulated webhook endpoint triggers a test event with dummy data that matches the specified event type schema.

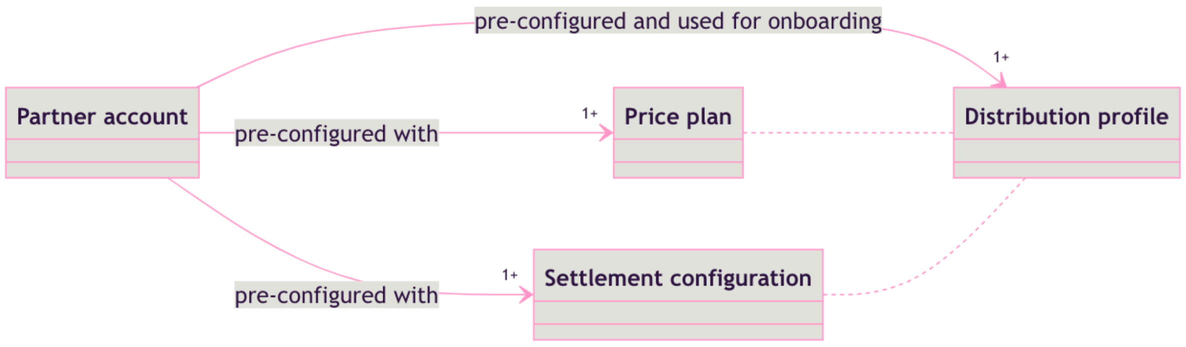
Example webhook event:

```javascript
JavaScript
{
  "metadata": {
    "event_type": "payment.request.state-change.authorized",
    "event_id": "d9f9b1a0-5b1a-4b0e-9b0a-9e9b1a0d5b1a",
    "event_version": "v1",
    "occurred_at": "2024-01-01T12:00:00Z",
    "correlation_id": "2d1557e8-17c3-466c-924a-bbc3e91c2a02",
    "account_id":
"krn:partner:product:payment:ad71bc48-8a07-4919-a2c1-103dba3fc918",
    "product_instance_id":
"krn:partner:product:payment:ad71bc48-8a07-4919-a2c1-103dba3fc918",
    "webhook_id":
"krn:partner:global:notification:webhook:120e5b7e-dee8-43ca-9858-dca726e639b5",
    "live": false
  },
  "payload": {
    "payment_request_id":
"krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
    "payment_reference": "partner-payref-1234",
    "merchant_reference": "order-5678",
    "state": "AUTHORIZED",
    "previous_state": "PENDING_CONFIRMATION",
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0"
  }
}
```

Step 3: Account configuration management

This section explains the available configurations for setting up accounts and provides recommendations on how to use them.

When partnering with Klarna as an integrator and distributor of our solutions, we will work together in the definition and your Partner Account will have access to certain resources that are assigned by Klarna, such as a price plan or settlement configurations.



You will be able to query these at any given point in time to review the definitions agreed via these endpoints

- `/payment/settlement-configurations`
- `/payment/pricing/price-plans`

In response you will receive a list of `settlement_configuration_id and price_plan_id.`

Release notes

📅 Further configuration points may become available with future releases.

Price plans

Price plans are read-only and cannot be modified through the APIs. They outline the pricing rates for transactions based on the Merchant Category Code (MCC) and market specifics.

Each price plan includes rates that may vary by market and channel type. These are created and maintained by Klarna.

Release notes

📅 Retrieval of Price Plans via the APIs will be available in future releases.

Settlement configuration

A settlement configuration outlines how Klarna will settle funds to a partner. This setup includes various components crucial for processing payouts:

- **Payout schedule**: Specifies when the payout will occur based on when the transaction is captured.
- **Settling business entities**: Identifies the legal entity receiving the funds. This is also the entity linked to the bank account where funds are deposited.
    - This generally refers to the local entity of the Acquiring Partner that onboarded the merchant.
- **Bank account details**: Information regarding where funds should be transferred.
- **Currency configuration**: Determines which entity receives payouts for which currency. See the cases below for more detail.
- **Payout prefix**: A defined prefix added to payouts to assist with identification and reconciliation.

As an integrator and distribution partner, when you integrate merchants through Klarna, you handle the settlements towards them, thereby simplifying the reconciliation process on their end. Our recommendation is that the Klarna account structure matches the account structure defined in your platform, keeping a 1:1 account ratio between the two systems.

Example

1. LocalPay is a global Acquiring Partner that processes transactions across multiple regions. It prefers to receive payouts for all transactions globally to the same bank accounts for all merchants, regardless of their onboarding country or entity. Therefore, LocalPay does not need to specify any specific settlement configurations during the merchant onboarding process.

2. In contrast, GloboPay is a global Acquiring Partner that also processes transactions across multiple regions. It prefers to settle payouts to different bank accounts by country and currency for administrative reasons. For example,
    a. GloboPay prefers payouts for merchants onboarded in the UK to be directed to BankAccount1 for all currencies, while
    b. payouts for merchants onboarded in the EU should be directed to BankAccount2 for all currencies.

   Consequently, GloboPay needs to specify the applicable settlement configurations during the merchant onboarding process.

Onboard and manage merchants

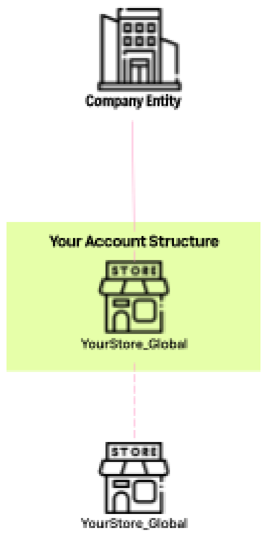
Step 1: Determine account structure

Partner accounts are used for managing the relationship between your merchants and Klarna's services. These accounts are created and managed by you using Klarna's Management API and contain critical information to facilitate a successful partnership. These blocks consist of:

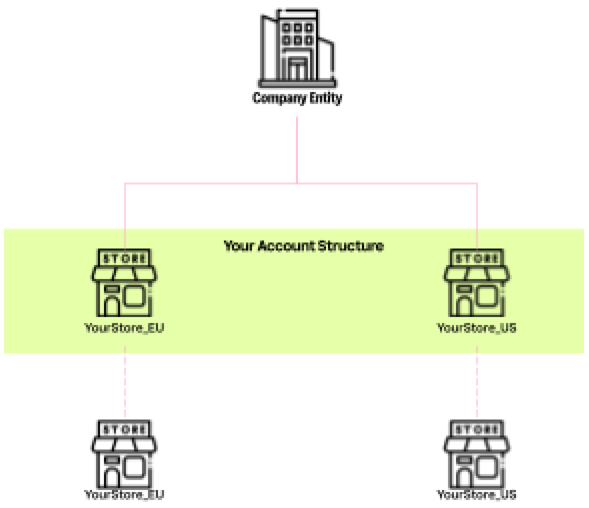
- **Account owner**: Information about the main representative of the merchant for Klarna.
- **Products**: List of Klarna products that are being used by the account.
- **Channels**: List containing the details of the website, mobile app or physical store where Klarna products are going to be made available
- **Extra account information**: Any information relevant for Klarna to efficiently run its fraud prevention functionality.

If a merchant operates through multiple entities, you as the integrator and distribution partner should **align the creation of the Klarna merchant account to the structure reflected in your own systems**.

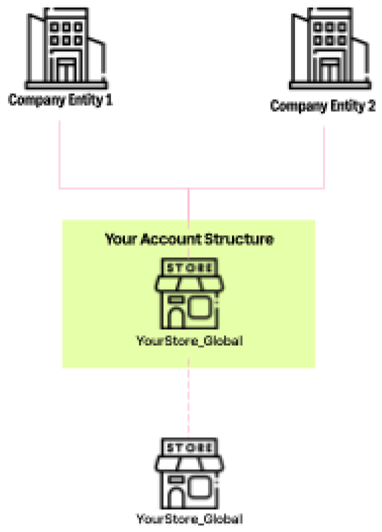
If a merchant uses a single entity for their partnership setup, this simplified structure should be mirrored in their Klarna account.



If a merchant uses multiple entities, resulting in different accounts generated in your systems, you should create multiple Klarna accounts for the merchant. Each account in your system should have a corresponding Klarna account.



If a merchant uses multiple entities globally but reflects them in a single account in your systems, you should maintain this structure by creating a single Klarna account for the merchant. Each account in your system should only have one corresponding Klarna account.



⚠️There is a limitation of one website channel per account, which means that multi-website or physical stores are not supported. More information on Channels are available in the Channel types and management section.

Release notes
📅 In future releases the channel object will express where the products from Klarna are going to be made available across multiple channels - including websites, physical stores or mobile apps.

Products and Accounts have independent life cycles, more information on product and merchant lifecycles are available in the Manage your merchant section.

Account structure use cases

Let’s take a look into a few different examples of how merchant accounts can be used to represent merchants:

Retail

In a typical retail setup, the structure includes a partner account, a payment product, and their channels.

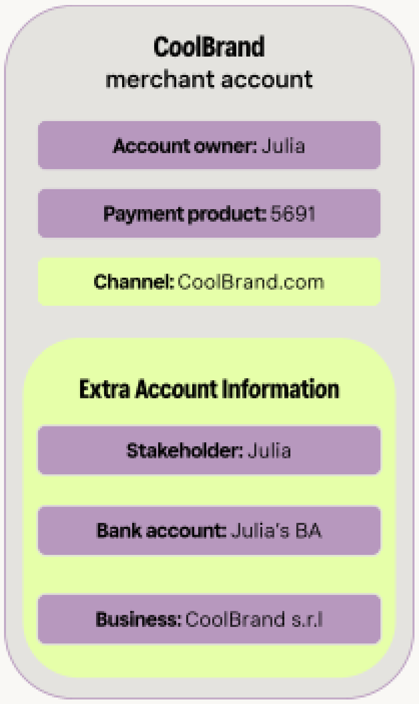
Example 1:

Consider a small clothing shop in Italy called "*CoolBrand*". CoolBrand consists of an online shop with the URL *Coolbrand.com*, it is owned by a single owner, Julia. The store operates under MCC 5691.

This store receives all their payouts from their Acquiring Partner in a single bank account. They operate under the same address to which they are registered in Milan.

This is an example of a basic partner account set up. It’s a single account, with a single payment product, and a single channel.

Given it’s 1:1:1, there is no need to identify anything when initiating a payment request for this merchant.



Onboarding Payload Example:

JavaScript
```
{
  "account_reference": "M123786123412",
  "account_name": "CoolBrand",
  "account_owner": {
    "given_name": "Julia",
    "family_name": "Doe",
    "email": "julia.doe@CoolBrand.com",
    "phone": "+15555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-33715
3c04a58",
      "type": "PAYMENT",
      "merchant_category_code": "5691"
    }
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://CoolBrand.com"
        ]
      }
    ]
  }
}
```

Multiple MCCs

Release notes
📅 This capability is currently not supported and will be available in future releases.

A merchant who has multiple Merchant Category Codes (MCCs) but operating under a single account on your side can be supported by adding payment products with different MCCs to a single account on Klarna's side through the Product object.

This model can be applied to merchants that sell different types of products or marketplaces. Details will be provided alongside future releases.

Multiple channels (websites, physical stores etc)

For companies with multiple websites, we allow you to model the channel configuration in the same way as it's modeled on your side. In case you require different accounts per channel on your end, you can apply the same model as the simple retail example to create all the different accounts.

In case you support multiple channels on the same account, the same can be created on our side.

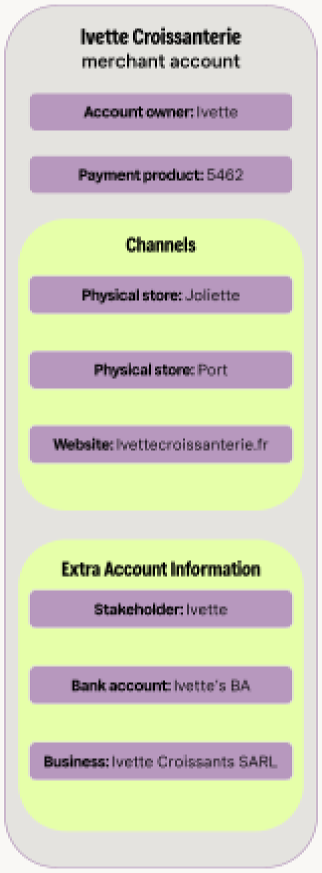
Example 3

Consider a pastry shop called "Ivette Croissanterie".

The pastry shop, owned by Ivette, has two physical stores located in Marseille, France. Ivette also sells pastries through an on-line store.

The account is configured with a payment product with the MCC 5462, with three channels. Two physical stores, and one website.

When initiating a payment request, the specific channel where the purchase is being made needs to be passed to the Partner Product API.



JavaScript
```
{
  "account_reference": "M123789922222",
  "account_name": "Ivette Croissanterie",
  "account_owner": {
    "given_name": "Ivette",
    "family_name": "Doe",
    "email": "Ivette@IvetteCroissanterie.fr",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-33715
3c04a58",
      "type": "PAYMENT",
      "merchant_category_code": "5462"
    },
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://IvetteCroissanterie.fr"
        ]
      }
    ],
    "physical-stores": [
      {
        "name": "Ivette Croissanterie Joliette",
      },
      {
        "name": "Ivette Croissanterie Port",
      }
    ]
  }
}
```

Release notes

📅 This capability is currently not supported and will be available in future releases.

Franchising

Franchise accounts should be modeled as close as possible to the business structure. Their Klarna account could be modeled either by

- managing all franchises via a single account
- creating individual accounts for each franchise owner

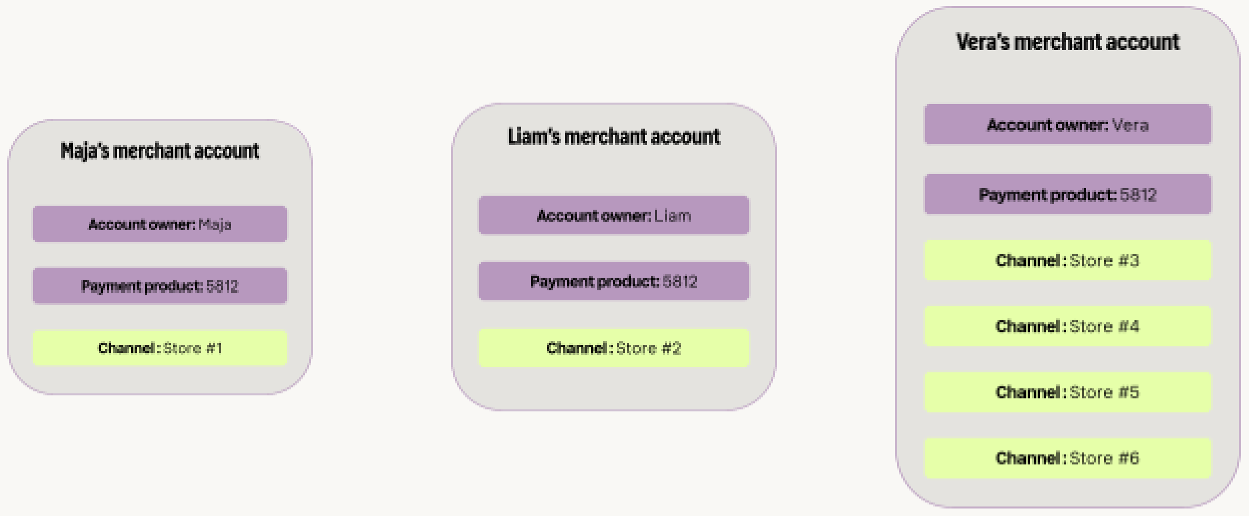
The form the Klarna account structure takes should be reflective of the structure of the business, but for most franchises a single account will suffice. This is dependent on how they wish to receive settlements.

Example 4

Consider a burger chain called William's Top Burger. William's operates under a franchise model, where currently they have three franchisees, owned by Maja, Liam and Vera. Out of the three, Vera was able to run a really successful business and now owns four different burger locations.

In this case, if the franchises are managed as separate accounts on your side and settlements are received separately from the Acquiring Partner, they can be modeled using a combination of the Retail and the Multi Channel examples, as below:

In the Partner Product API, the specific account ID and channel ID must be sent as part of the payload so Klarna can properly identify in which specific store the payment request is coming from.



Onboarding Payload Example:
To onboard, we would need three /onboard calls:

1st:

```JavaScript
{
  "account_reference": "M123789922222",
  "account_name": "Maja WTB",
  "account_owner": {
    "given_name": "Maja",
    "family_name": "Doe",
    "email": "maja.franchisee@wtb.se",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a
58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    },
  ],
  "channel": {
    "physical-stores": [
      {
        "name": "Maja WTB",
      },
    ]
  }
}
```

2nd:

JavaScript

```
{
  "account_reference": "M123789922222",
  "account_name": "Liam WTB",
  "account_owner": {
    "given_name": "Liam",
    "family_name": "Doe",
    "email": "liam.franchisee@wtb.se",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a
58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    },
  ],
  "channel": {
    "physical-stores": [
      {
        "name": "Liam WTB",
      },
    ]
  }
}
```

3rd:

JavaScript

```

{
  "account_reference": "M123789922222",
  "account_name": "Vera WTB",
  "account_owner": {
    "given_name": "Vera",
    "family_name": "Doe",
    "email": "vera.franchisee@wtb.se",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a
58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    },
  ],
  "channel": {
    "physical-stores": [
        "name": "Vera WTB #1",
      },
        {
          "name": "Vera WTB #2",
        },
        {
          "name": "Vera WTB #3",
        },
        {
          "name": "Vera WTB #4",
        },
      ]
    }
  }
```

Release notes

📅 This feature will be available in future releases.

Step 2: Onboard merchants

To start offering Klarna services to your merchants, you must first onboard them to Klarna.

When onboarding a merchant, make sure that you have defined the proper structure based on the merchant business model, follow the guidance in Step 1: Determine account structure section.

During the onboarding process, Klarna will assess the product object provided to enable the appropriate services. For example, selecting a payment product will activate all related payment-related services in all markets where Klarna is available, allowing a merchant to offer Klarna as a payment method.

While only a subset of data points is mandatory to complete this phase, we require that all relevant information available is provided to allow us to accurately manage potential risks and make informed decisions regarding merchants and transactions.

When a merchant is onboarded with Klarna, Klarna will create an account for the merchant and return an `account_id` in the onboarding response. At this point - they are ready to begin transacting in all markets where Klarna is available.

⚠️ Remember to store the **account_id** associated with the merchant as it is a required parameter for the integration of the Partner Product API.

The following example is only for illustrative purposes, and required parameters for onboarding may differ according to your commercial agreement and the parameters available to you.

Consult the **API reference** for a complete description of the request body parameters.

Request example:

```
Unset
{
  "account_reference": "M123786123412",
  "account_name": "John Doe Stakehouse",
  "account_owner": {
    "given_name": "John",
    "family_name": "Doe",
    "email": "john.doe@example.com",
    "phone": "+18445527621"
  },
  "products": [
    {
      "type": "PAYMENTS",
      "merchant_category_code": "7995"
    }
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://example.com"
        ]
      }
    ]
  }
}
```

Response example:

```
Unset
{
  "account_id": "krn:partner:global:account:live:LWT2XJSW"
}
```

Handling rejected onboardings

Integrating Klarna payment solutions requires accurate and complete merchant data to ensure seamless transaction processes and fraud prevention. This section covers the essential steps and protocols activated when discrepancies in merchant data are detected during the onboarding process

Identification of incomplete or incorrect data

Klarna's onboarding API performs real-time validations on all incoming data. When data fields are missing or incorrect, the system triggers an automated response outlining the specific issues. These validations cover critical information such as business details, tax IDs, and contact information.

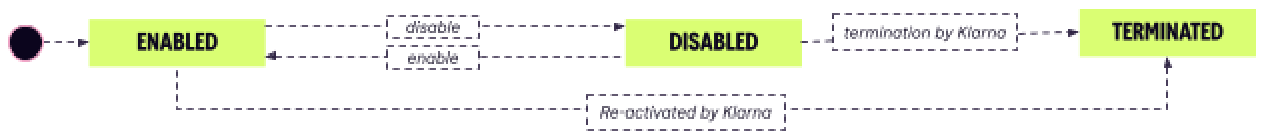
Step 3: Manage your merchant payment products

Using the Management API, you have full control over merchant accounts and their associated data. Each data object within an account is accessible via standard REST endpoints, allowing you to update, create, and delete resources as needed.

Through the Management API, you have full control over merchant accounts and their associated data. Each data object within an account is accessible via standard REST endpoints, allowing you to update, create, and delete resources as needed.

Products within these accounts follow a specific lifecycle, ensuring they are managed efficiently from inception to discontinuation. The lifecycle diagram below provides detailed insight into these stages.

Familiarize yourself with these processes and actively use the available tools to optimize account operations and product handling. Review the lifecycle diagram to better understand each stage.



Where:

Status	Definition
ENABLED	the product is capable of processing new payment requests
DISABLED	the product **cannot** process new payment requests. It can be reverted if you are the one that disabled it. Other operations, such as post-purchase calls, may still succeed.
TERMINATED	The payment product is fully disabled by Klarna due to fraud.

These statuses can be manipulated through the following APIs:

- Disable: `/v1/accounts/{account_id}/products/payment/disable`
- Enable: `/v1/accounts/{account_id}/products/payment/enable`

Release notes

📅 *DISABLE state for Payment Products will be supported in later releases.*

The following table lists all different events supported by Klarna webhooks for product management that will allow you to get immediate notification when certain events take place, in order to act on them immediately.

Use Case	When	Event name
Product disabled webhook	A product is disabled due to any reason, including suspension due to fraudulent behavior	`partner.account.product.payment.state-change.disabled`
Product enabled webhook	A product is enabled due to any reason, including recovery from fraud	`partner.account.product.payment.state-change.enabled`

Consult the API reference for a complete description of the request body parameters.

⚠️Klarna may also disable an account based on fraud or other unsavory behavior.

Channel types and management

A key component of a partner account is the channel. Channels represent where Klarna's services are going to be available to shoppers via the merchant. Channels can be one of the following types:

Types	Definition
Website	Online channel where Klarna's payment products are shown
Physical store	Brick and mortar store where a shopper can use Klarna to pay for goods
Mobile app	Mobile app where Klarna's payment options can be used

⚠️Each partner account must have at least one channel. If an account has only one channel, no further specifications are required.

Release Notes

📅 Future releases will support multiple channels within a single account.

When a partner account contains multiple channels, it is essential to specify the exact channel through which a payment is being processed at the time of making a payment request. This ensures that the correct brand identity features are displayed to shoppers, enhancing their shopping experience. Accurately identifying the payment channel is also crucial for the effectiveness of Klarna's fraud assessment systems.

Channel collections

Channel collections are designed to unify brand identity across various shopper touch points. Each channel collection comprises merchant-specific details such as branding, support channels, and social media links. This information is crucial for enhancing the shopper's purchase and post-purchase experience, for example in the Klarna app.

To efficiently manage branding across multiple channels, channel collections use the `collection_reference` attribute. Once a collection is created and assigned a reference, this reference can be assigned to different channels to apply consistent branding without the need to replicate branding details for each channel individually.

To implement channel collections in your system, start by defining the branding, support, and social media details within a collection. Assign a unique `collection_reference` to this collection, and apply this reference to each desired channel. This approach ensures that the merchant's brand identity is consistently represented, enhancing recognition and trust among shoppers.

Consult the **API reference** for a complete description of the request body parameters.

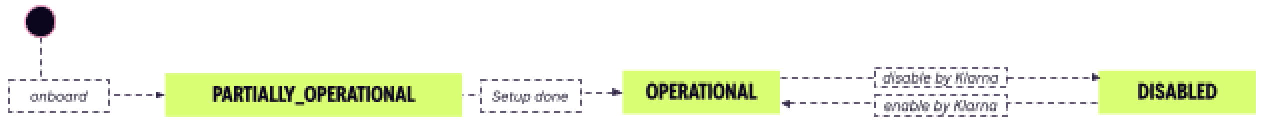
Keep merchant data updated

Data integrity is essential

Maintaining accurate merchant account information is essential for the seamless operation of Klarna's services. It is crucial to update this information whenever there are any changes in your systems to a merchant's account details.

Account lifecycle webhooks

A Partner Account has a simplified life cycle, as mostly just implementing logic around the Product life cycle is sufficient for a successful integration with our APIs. The account lifecycle **can only be manipulated by Klarna** and is represented in the diagram below.



Where:

Parameter	Definition
PARTIALLY_OPERATIONAL	Represents an account that was just onboarded and is currently being set up in our systems. **The account can process transactions**, but can not yet be managed. This state is transitional, and typically moves to OPERATIONAL within seconds. y.

Parameter	Definition
OPERATIONAL	Represents an account that is fully operational and can both be managed and process transactions.
DISABLED	Represents an account that is no longer operational in any respect. The account cannot be managed by the partner or have any new products added to it. All products within an account are fully disabled as a part of this status.

Ensure that you configure management webhooks to receive notifications about status changes and merchant account statuses.

The following table lists all different events supported by Klarna webhooks for account management that will allow you to get immediate notification when certain events take place, in order to act on them immediately.

Use case	When	Event name
Account fully onboarded webhooks	An account is fully set up on Klarna's side	`partner.account.state-change.operational`

The following example reflects the payload structure for `partner.account.state-change.operational` event. It differs slightly from the general webhook format as it doesn't contain a product_instance_id, given this event affects the whole account.

Example:

```
Unset
{
  "metadata": {
    "event_type": "partner.account.state-change.operational",
    "event_version": "v1",
    "occurred_at": "2024-01-01T12:00:00Z",
    "correlation_id": "2d1557e8-17c3-466c-924a-bbc3e91c2a02",
    "account_id": "krn:partner:account:live:2AIMNWR6IYZVD",
    "webhook_id":
"krn:partner:global:notification:webhook:120e5b7e-abcd-4def-8a90-dca726e639b5",
    "live": true
  },
  "payload": {
    "account_reference": "M123786123412",
    "state": "OPERATIONAL",
    "previous_state": "PARTIALLY_OPERATIONAL"
  }
}
```

Enable interoperability of Klarna products

Interoperability refers to the ability of Klarna's products to work seamlessly regardless of integration method. This seamless interaction ensures merchants can provide a smooth shopping experience using Klarna's features, such as Express checkout, On-site messaging, and Sign in with Klarna while using an Acquiring Partner to add Klarna to their payment processing.

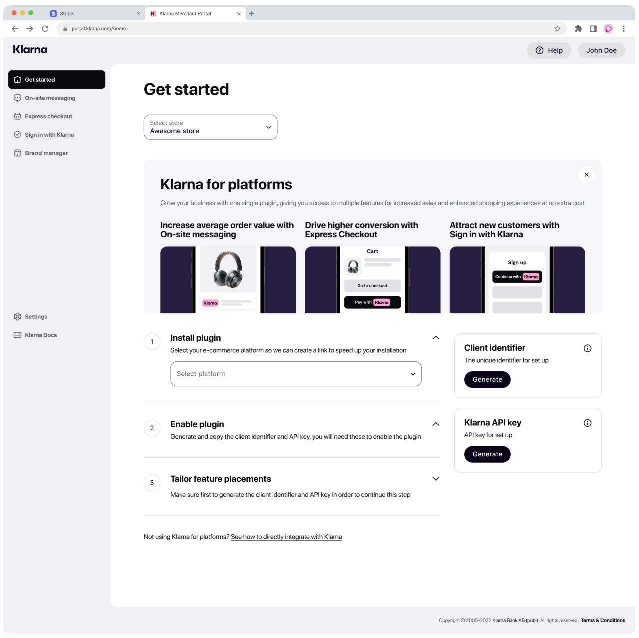
The objective is to guarantee that Klarna's products are uniformly available across all integrations, ensuring that there are no gaps in the merchant offering - whether directly integrated with Klarna or through any partner. This approach not only streamlines operations for our partners but also guarantees that shoppers benefit from seamless transactions and features, regardless of the merchants integration method.

Step 1: Grant access to Klarna's ecosystem

As Klarna distributes non-payment products directly to merchants, such as On-site messaging. It is essential for merchants to access the Klarna ecosystem after a successful authentication.

This can be achieved through one or more of the following solutions:

- **Deep Link integration** (recommended): incorporate a deep link to the Klarna portal within your dashboard. Ensure this link is clearly labeled to indicate its purpose and integrates smoothly with your existing user interface.
- **E-mail access provision** (alternative) : if your ecosystem lacks an admin area for granting access through a deep-link, you will need to use Klarna's onboarding API to grant your accounts access to the portal. Access to the portal is given by enrolling a user using their email address.



Get started

Select store
Awesome store

Klarna for platforms

Grow your business with one single plugin, giving you access to multiple features for increased sales and enhanced shopping experiences at no extra cost

Increase average order value with On-site messaging

Drive higher conversion with Express Checkout

Attract new customers with Sign in with Klarna

1 Install plugin
Select your e-commerce platform so we can create a link to speed up your installation
Select platform

2 Enable plugin
Generate and copy the client identifier and API key, you will need these to enable the plugin

3 Tailor feature placements
Make sure first to generate the client identifier and API key in order to continue this step

Client identifier
The unique identifier for set up

Klarna API key
API key for set up

Not using Klarna for platforms? See how to directly integrate with Klarna

Copyright © 2005-2022 Klarna Bank AB (publ). All rights reserved. Terms & Conditions

Release Notes

📅 In a later release an endpoint will be introduced to allow the creation of Merchant Portal accounts.

Through the portal, merchants are able to access and manage Klarna boost products. They have no ability to access or manage any payment related processes, which happens through the partners platform. The core functionalities available through the Klarna portal include:

- **Setup of Klarna visual assets**: merchants can enhance how their brand is presented in the Klarna app by adding branding elements such as logos, icons, social media URLs, and display names. These adjustments contribute to a smoother and more engaging self-serve shopper experience.
- **Boost product access**: access to Boost products like On-site messaging, Klarna Express checkout, and Sign in with Klarna.
- **Merchant credential management**: merchants can generate their own client identifiers and API keys. Credentials generated by merchants onboarded by an Acquiring Partner are limited and do not have permission to access or manage payment processing.
- **Merchant campaigns**: additionally, the Klarna portal enables merchants to launch campaigns offering 0% financing and other promotions.

In specific instances based on available features in your back office system, a merchant will be able to access additional capabilities via the Klarna portal, such as dispute management.

This access is assigned through the use of the `roles` array outlined below. Individual `roles` are defined by Klarna during your onboarding process.

Creating a deep link

Integrating deep linking within your ecosystem ensures your merchants are able to manage Klarna products from within your ecosystem without requiring additional login credentials - ensuring the benefits of direct integration while retaining the simplicity of your value proposition.

You may only create deep links for the merchant accounts that you have previously created through a POST request to `/v1/accounts/{account_id}/portal/deep-links`.

A deep link to the Klarna portal can be requested via API, which is a one-time use link that expires after 60 seconds. Once the user has accessed the Klarna portal using the deep link, no additional authentication will be needed and the session initiated will be valid for 8 hours, after which the user will be logged off and will need to log in again.

At this moment you can define the level of access granted to the user by setting the **`roles`** array.

Parameter	Definition
`user_reference`	The identifier of the user whom the deep-link is generated for. It should be traceable for auditing, support or debugging purposes and it should not be a personal identifier such as email. This reference is provided by the Partner.
roles[]	Contains the permissions to apps inside the Merchant Portal. Roles to be defined at a later date.

Release notes

📅 `roles` will be defined in later releases.

Upon creating a deep-link successfully, Klarna will provide a URL that allows access to the Klarna portal. This link should be requested by you, as a distribution partner, only after the merchant expresses a desire to enter the Klarna portal. The URL will be issued only if the partner account credentials are verified by Klarna and match the partner authorized to manage the specified account.

Step 2: Consume and pass key identifiers

To provide the best performing shopper experience, it is crucial for all parties within the Klarna network to effectively manage and utilize key identifiers provided by Klarna. These identifiers, specifically the `shopping_session_token` and `payment_confirmation_token`, are integral to ensuring a seamless, secure, and efficient transaction process across various platforms and integration types. Understanding and implementing these identifiers correctly can significantly contribute to a more personalized and cohesive shopper experience.

Shopping session token

The `shopping_session_token` is a cornerstone identifier within Klarna's ecosystem, enhancing the shopper experience by providing continuity across integrations with different Klarna services such as On-site Messaging and Sign in with Klarna. It minimizes friction for shoppers by maintaining a consistent session across different platforms and providing a framework for personalization and keeping shoppers signed in across services.

The `shopping_session_token` is provided directly to the party implementing the Klarna Web SDK. Merchants integrating Klarna Boost products directly will receive the identifier directly from Klarna and must later provide it to Klarna via your services. If you manage the client-side presentation of Klarna, you may receive the `shopping_session_token` directly and must make it available to the merchant so that it can be used across all Klarna services.

Implementation of the `shopping_session_token` allows for key operational improvements like:

- **Increased flexibility** across Klarna services, ensuring that shoppers can enjoy a seamless transition between different merchant offerings without repeated logins or data entries.
- **Enhanced security and fraud detection** by providing a broader understanding of customer activity, a consistent identifier helps in monitoring and mitigating potential fraud across different sessions and platforms.

To optimize the use of `shopping_session_token`, you, as a distribution partner, must adhere to the following requirements:

- **Integration**: merchants should have the option to include a shopping session token, specifically namespaced for Klarna, when integrating any Partner product that utilizes Klarna services. This parameter is optional and its absence should not cause technical issues.
- **Parameter handling**: when a merchant provides this optional identifier through the API, it is the responsibility of the acquiring partner to:
    - Reuse the identifier and return it to Klarna using the Web SDK.
    - Include the identifier when creating a server-side payment request.
    - Return the identifier to the merchant if using Klarna Web SDK, as appropriate.
- **Visibility and usage**: when Klarna provides you with a shopping session token, it must be made visible to the merchant. This enables the merchant to use the identifier across different products or integrations effectively.

Key touchpoints

- **Web SDK integration**: if Web SDK is being used, the `shopping_session_token` should be incorporated to maintain session continuity. This ensures that the shopper can navigate through different stages of interaction without session breaks, enhancing user experience and maintaining session integrity.
- **Upstream messaging**: if upstream messaging is being used, integrating the `shopping_session_token` can help tailor the messages based on the shopper's previous interactions and behaviors across the Klarna network, providing a personalized and cohesive shopping experience.
- **Payment-specific calls**: during any payment-specific calls, including the `shopping_session_token` helps link the transaction to a specific shopper session. This is crucial for tracking the shopper's journey through different payment stages and for supporting advanced security measures like fraud detection.

By integrating and effectively managing the `shopping_session_token`, you and your merchants can provide a smart and more secure shopping experience, fostering greater shopper loyalty and operational efficiency.

Release notes

📅 `shopping_session_token` will be available in future releases.

Payment Confirmation Token

The `payment_confirmation_token` is generated at the completion of a transaction using Klarna services. This token is crucial for confirming and finalizing transactions within your system, ensuring that the payment process adheres to both Klarna and your protocols.

Implementation of the `payment_confirmation_token` allows the merchant to benefit from Klarna Express checkout, allowing your partners to benefit from Klarna's express checkout in addition to your integrated Klarna services. The payment_confirmation_token is provided by Klarna in response to a successfully completed express checkout transaction. Enabling merchants to pass this token via your services allows for seamless integration with Klarna's Express Checkout.

To effectively integrate and utilize the `payment_confirmation_token`, adhere to the following streamlined process:

- **Endpoint requirements**: establish at least one endpoint that allows merchants to complete a payment request using a Klarna confirmation token as a parameter. This endpoint, whether existing or new, should require essential payment details such as payment_amount and currency, but avoid any additional data requirements that could complicate integration.
- **Payment confirmation**: utilize the confirmation token to confirm the payment via Klarna's Partner product API.
- **Payment lifecycle management:** once the payment is registered, it should follow the standard lifecycle of the acquiring partner's system. The payment should be managed and accessible in the same manner as payments integrated directly without Klarna Express checkout.
- **Seamless integration:** ensure there is no distinction in handling or behavior between transactions completed using Klarna Express checkout and those placed directly through your system. As the acquiring partner, you must confirm payments from Klarna using the same protocols as for direct integrations.

This approach guarantees minimal integration effort while maintaining consistency in payment processing and user experience.

Key touchpoints

To streamline the payment process, the integration of a payment confirmation token should be made available at any touchpoint which may result in the completion of a transaction for other payment methods. Here are the key touchpoints where the payment confirmation token may play a crucial role:

- **Create calls:** during the initiation of any `create_payment_request` or `confirm_payment_request` calls, implementing the payment confirmation token can simplify the merchant's integration process. This setup streamlines the transaction flow by securely encapsulating payment details.
- **Tokenization or off-session flow:** utilizing any tokenized payment flow is an excellent opportunity to introduce the payment confirmation token. This approach is beneficial as merchants anticipate a completed transaction from the existing integration, enhancing the security and efficiency of the payment process.
- **Express checkout flows:** For any alternative express checkout options provided, incorporating the handling of the payment confirmation token is advisable. This ensures that the Express checkout process aligns well with secure transaction practices.

The successful completion of a transaction using the payment confirmation token triggers the same notifications and status updates as seen in standard Klarna payment processes. It is crucial for your integration to listen to Klarna webhooks as a reliable source for transaction status updates. Always ensure to validate and test these flows thoroughly to maintain seamless and secure operations.

Release notes

📅 `payment_confirmation_token` will be available in future releases.

Step 3: Supplementary shopping data

To ensure a seamless payment experience and enhance the data richness of transactions, it's crucial to transmit any supplementary shopping data collected to Klarna. This process utilizes the `merchant_supplementary_data` parameter present in the Partner product API, enriching the quality and detail of the information available for each transaction.

To facilitate the provision of additional data helpful to Klarna's decision making, a passthrough field must be made available to merchants. This field supports the inclusion of information required by Klarna that may not be initially collected. Adherence to the following principles is crucial:

- **Data integrity**: The data within the passthrough field must not be altered or validated beyond the constraints outlined in the integration guidelines.
- **Data transmission:** If the merchant utilizes the supplementary data, the acquiring partner should relay this information through the payments feature of the Partner Product API.
- **Efficiency in data handling**: Any supplementary data already collected by you, as partner, should be transmitted to Klarna without requiring additional efforts from the merchant. If no supplementary data is gathered by you, a passthrough field allowing partners to pass required information must be made available.

These measures ensure that merchants can add valuable data to their payment requests seamlessly and without unnecessary complexity.

Key touchpoints

To ensure comprehensive integration, it is essential that supplementary shopping data is accessible to the merchant during all payment-related interactions. This integration may occur at various stages, depending on your's system setup:

- **Initiation of SDKs**: When initiating the Klarna SDK, ensure that supplementary shopping data fields are incorporated. This step is crucial for capturing relevant data right from the start of the transaction process.
- **Transactional API calls**: During any API calls made to Klarna, supplementary shopping data should be included. This ensures that all transaction-related information conveyed to Klarna is comprehensive and detailed.

Incorporating this field across all touchpoints is vital for maintaining data continuity and enhancing the payment experience. By doing so, merchants can leverage this data to facilitate smoother transactions and improve overall shopper satisfaction.

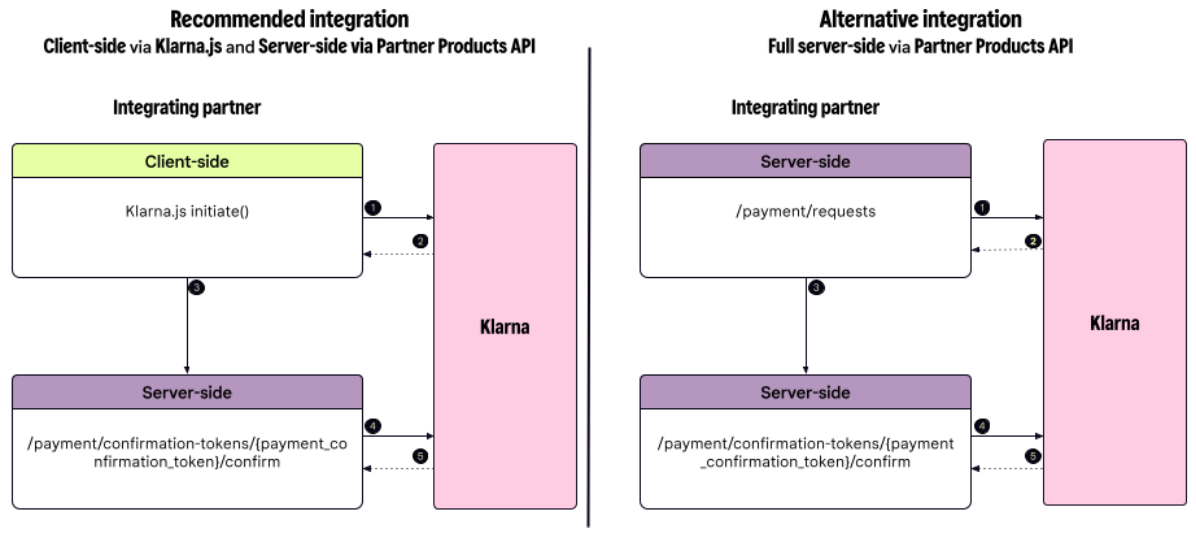
Release notes

📅 Supplementary data support will be available in future releases.

Processing of Klarna payments

To integrate Klarna payment solutions as an Acquiring Partner, we provide flexible integration options to accommodate various integration styles. These options allow you to tailor the integration to your specific needs and the needs of your merchants.

- For offerings where you, the acquiring partner, own a client-side component—such as hosted checkout solutions or embedded components—we require a client-side integration leveraging Klarna's SDK, Klarna.js, and Klarna's Partner Product API. This ensures a seamless implementation of Klarna payments directly on your platform, providing advanced functionality for security and ease of checkout.
- For offerings that solely provide server-side integration to merchants, where the merchant owns the client-side integration of Klarna, you should adopt the Partner Product API and expose redirect links to your merchants. This ensures compliance while allowing for flexibility to accommodate your integration style.


Recommended integration
Client-side via Klarna.js and Server-side via Partner Products API
Integrating partner
Client-side
Klarna.js initiate()
Server-side
/payment/confirmation-tokens/{payment_confirmation_token}/confirm
Klarna
Alternative integration
Full server-side via Partner Products API
Integrating partner
Server-side
/payment/requests
Server-side
/payment/confirmation-tokens/{payment_confirmation_token}/confirm
Klarna

1. Initiate the payment request client-side using the `Klarna.js initiate()` method.
2. Retrieve a payment confirmation token from a successful checkout.
3. Confirm the payment server-side to obtain an authorized transaction.

1. Initiate the payment request server-side and redirect the shopper to a Klarna Payment page.
2. Retrieve a payment confirmation token from a successful checkout.
3. Confirm the payment server-side to obtain an authorized transaction.

Use the `shopping_session_token` to create a unified and seamless shopping experience, especially when your integration includes both client-side and server-side elements. This id is provided by the Klarna web SDK in response to any initiating action where an existing `shopping_session_token` is not provided. By providing this parameter to your merchants (where the initiation of Web SDK is owned by you), and allowing the merchant to pass the parameter in all your requests, you tie together these elements, ensuring consistent data flow and maintaining session continuity throughout the shopper's journey.

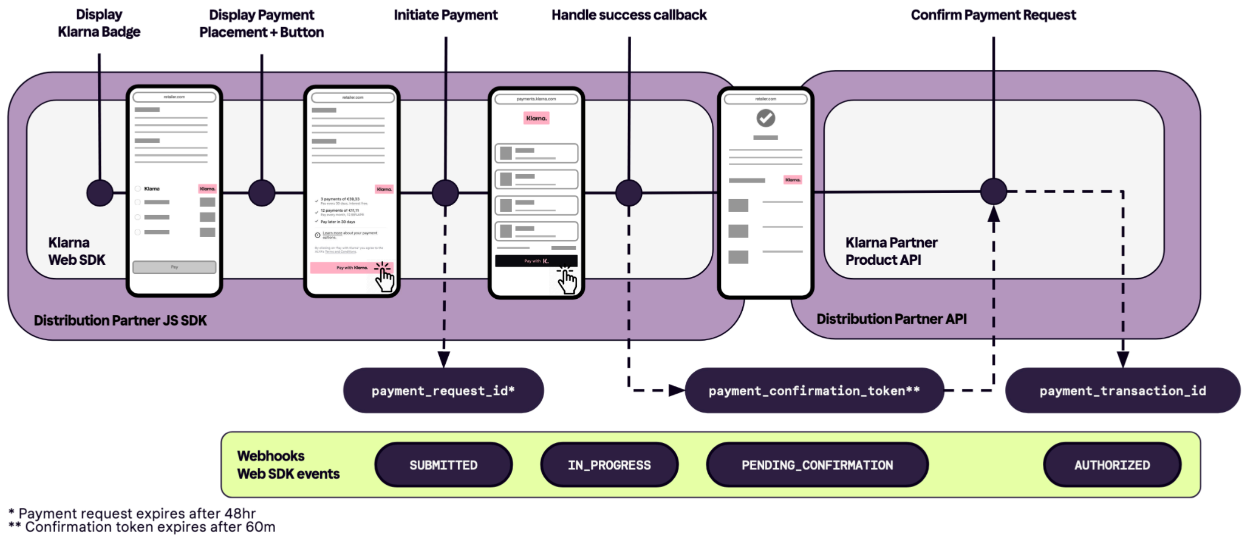
Implementing the `shopping_session_token` enhances personalization, security, and cohesion in the checkout process, reducing friction and leading to higher customer satisfaction and increased conversion rates. More information on leveraging `shopping_session_token` to improve the customer experience is detailed in the interoperability section.

Web ecommerce transaction

Recommended integration: Payment request initiated via Klarna.js

A client-side integration via the Javascript library **Klarna.js** provides shoppers with a seamless checkout experience allowing them to complete their purchase without leaving the merchant's website. Furthermore, it allows additional features to enrich the shopping experience, such as our 1-click express checkout solution, Klarna Express checkout.

Below is an illustration of the integration flow:



Step 1: Include the Klarna.js

To ensure optimal performance and security, include the Klarna.js script on every page of your website that displays a Klarna component. Always load the script directly from https://js.klarna.com/web-sdk/v1/klarna.js; do not include it in a bundle or host it yourself. This approach guarantees that you are using an up to date version of the script, thereby enhancing security and enabling automatic updates for new features.

When Klarna.js is initiated, the `shopping_session_token` will be provided which should be handled as outlined in the shopping session token section. Within your create session APIs, you should allow the merchant to provide a `shopping_session_token` in case they have previously started a Klarna session regarding the customer arriving at the checkout.

Initialization of the Web SDK

```javascript
<script defer
       src="https://js.klarna.com/web-sdk/v1/klarna.js"
       data-client-instance-name="IntegratorName"
></script>

<script>
  window.KlarnaSDKCallback_IntegratorName = async (Klarna) => {
    // Klarna SDK ready to be initialized with klarna.init(...)
    const klarna = await Klarna.init({
      accountId: "<the partner account id>",
      clientId: "<your client id>",
    })
    // Klarna SDK ready to be utilized
  }
</script>
```

See supported browsers consideration by the Klarna Web SDK here. Ensure that your usage of the SDK aligns with these browsers to guarantee full functionality.

⚠️ Klarna requires acquiring partners to use the programmatic initiation of the script, removing reliance on the global callback to understand when Klarna is ready and ensuring merchant-initiated SDKs do not cause conflicts in the loading of the Klarna Web SDK.

Therefore, the client instance name must be passed via the script tag, as it is used to infer the name of the callback that the PSP should use to access their Klarna object (e.g., `KlarnaSDKCallback_IntegratorName`).

Step 2: Check availability and display Klarna at the checkout

As part of integrating Klarna payment solutions into your offerings as an Acquiring partner, we require that a structured approach is followed to ensure a seamless implementation. Central to this is confirming the availability of Klarna payment options and customizing the payment messaging to enhance the user experience.

By utilizing methods such as `canMakePayment()` and `getPlacementContent()`, you can ensure that your platform effectively offers Klarna's payment solutions. The objective is to achieve an optimal checkout process, leading to increased customer satisfaction and higher conversion rates.

2.1 - Confirm availability

Invoke the `canMakePayment()`method to find out if a payment is currently possible. This method checks if Klarna is available for a specific combination of customer country, currency, and amount. This allows Klarna to roll out to new markets without requiring direct communication with partners, and ensures that your platform can dynamically adapt to Karna's availability.

JavaScript
```javascript
const canMakePayment = await klarna.Payment.canMakePayment({
  currency: "EUR",
  country: "DE",
  paymentAmount: 10000,
})

if (canMakePayment) {
  // Show Klarna in the checkout
}
```

The `canMakePayment()` method returns a promise which resolves to a boolean value (true or false):

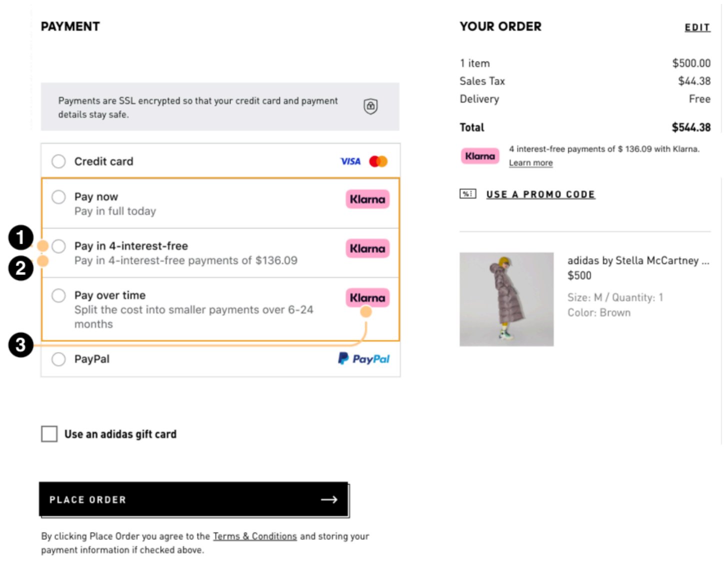
- If `true`, it indicates that the payment can be processed using Klarna and you should proceed with the payment process as planned.
- If `false`, Klarna is not available for the specified combination of parameters, and you should provide an alternative payment method to ensure a smooth checkout experience for your customers.

2.2 - Get content for Klarna's payment badge, descriptor, and subheaders

1. Payment descriptor
High-level call to action. Provided in Klarna response.

2. Payment subheader
More detailed breakdown of the value of the option presented. Provided in Klarna response.

3. Klarna badge
Klarna logo. Provided in Klarna response.



PAYMENT

Payments are SSL encrypted so that your credit card and payment details stay safe.

- Credit card
- Pay now — Pay in full today — Klarna
- Pay in 4-interest-free — Pay in 4-interest-free payments of $136.09 — Klarna
- Pay over time — Split the cost into smaller payments over 6-24 months — Klarna
- PayPal

Use an adidas gift card

PLACE ORDER

By clicking Place Order you agree to the Terms & Conditions and storing your payment information if checked above.

YOUR ORDER EDIT

1 item	$500.00
Sales Tax	$44.38
Delivery	Free
Total	**$544.38**

Klarna 4 interest-free payments of $ 136.09 with Klarna. Learn more

USE A PROMO CODE

adidas by Stella McCartney ...
$500
Size: M / Quantity: 1
Color: Brown

By using the `getPlacementContent()` method of klarna.js `Messaging` package it is possible to retrieve information about descriptors, subheaders and the Klarna badge.

Release notes

📅 The `getPlacementContent()` method will be made available in a future release.

To achieve a flexible and global solution that allows merchants to choose their preferred presentation method, as per Enhance the shopping experience, we require you to **allow your merchants to pass the `message_preference` parameter at checkout**. If no preference is provided by your merchant, the default behavior is to display all available options separately.

Parameters

Parameter	Definition
`locale`	Locale to use for returned content. BCP 47 (concatenation of language code (ISO 639-1 format) + "-" + country code (ISO 3166-1 alpha-2 format)) Example: en-US
`currency (optional)`	The purchase currency of the transaction. Formatted according to ISO 4217 standard, e.g. USD, EUR, SEK, GBP, etc. If not provided, default currency for the locale is used.
`payment_amount`	Total amount of a one-off purchase, including tax and any available discounts. The value should be in non-negative minor units. Eg: 25 Dollars should be 2500.
`message_preference (optional)`	Indicates the preferred payment option. Enum: ● PAY_NOW ● PAY_LATER ● PAY_OVER_TIME ● KLARNA

Example

Possible scenarios for `message_preference` parameter:

- If no `message_preference` is specified, the messaging copy for the **all-option approach** will be returned by default (default and recommended solution)
- If `message_preference`=KLARNA is specified, Klarna will return the messaging copy for the **Single-option approach**

If neither dynamic option is supported, the static one-option presentation of Klarna must be leveraged. See more information regarding the available approaches here.

Request example:

```
JavaScript
const descriptor = klarna.Messaging.getPlacementContent({
    key: "payment-descriptors",
    payment_amount: 20000,
    locale: "fr-CA"
})
```

In the response, Klarna will return an array of payment descriptors listing the content to be displayed (`PAYMENT_DESCRIPTOR`, `PAYMENT_DESCRIPTOR_SUBHEADER`, `KLARNA_BADGE`), indexed per `payment_option_id`.

The `payment_option_id` is a parameter that should be used when initiating the payment request, as it allows you, as the acquiring partner, to preselect the corresponding payment option when the shopper enters the purchase flow.

Release notes

📅 PAYMENT_DESCRIPTOR_SUBHEADER `support` will be added in a later release.

Response example:

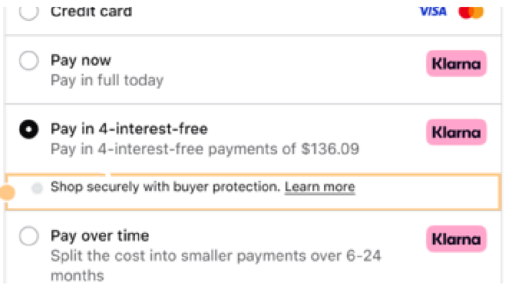
```
Unset
{
  "paymentDescriptors": [
    {
      "paymentOptionId": "123xyz",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Pay now"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Pay in full today"
          },
          {
            "name": "KLARNA_BADGE",
            "type": "IMAGE",
            "url": "...",
            "label": "Klarna logo"
          }
        ]
      }
    },
    {
      "paymentOptionId": "123xyz",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Pay in 4-interest-free"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Pay in 4-interest-free payments of $25.00"
          },
          {
            "name": "KLARNA_BADGE",
            "type": "IMAGE",
            "url": "...",
            "label": "Klarna logo"
          }
        ]
      }
    },
    {
      "paymentOptionId": "123xyz",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Pay over time"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Split the cost into smaller payments over 6-24 months"
          },
          {
            "name": "KLARNA_BADGE",
            "type": "IMAGE",
            "url": "...",
            "label": "Klarna logo"
          }
        ]
      }
    }
  ]
}
```

2.3 - Display the Klarna checkout placement

The messaging package contains a custom element that can resolve into a collection of what we denote “placements''. These placements represent a form of visual text or imagery that represents the Klarna brand.

Checkout Placement

Displayed when the customer selects a Klarna payment option, this section includes a buyer protection USP and a "learn more" link.


Credit card
VISA
Pay now
Pay in full today
Klarna
Pay in 4-interest-free
Pay in 4-interest-free payments of $136.09
Klarna
Shop securely with buyer protection. Learn more
Pay over time
Split the cost into smaller payments over 6-24 months
Klarna

JavaScript
```javascript
var klarnaMessaging = klarna.Messaging.placement({
  key: "checkout",
  amount: 10000,
  currency: "USD",
  locale: "en-US"
}).mount("#checkout");
```

Static assets

If relying on external dependencies is not feasible when building the checkout, or if you, the acquiring partner, do not own the client-side presentation of Klarna and cannot accept the `payment_option_id`, all assets (such as payment descriptors, payment subheaders, and the Klarna badge) are accessible as static content on docs.klarna.com. Klarna requires that up-to-date branding be in place in all checkouts, and it is the responsibility of the integrator to ensure that static assets remain current.

When using static assets, only the "one option" payment presentation will be available. Attempting to present multiple options will add friction, as any preselection made in your checkout will not persist within the Klarna session.

⚠️ Klarna may periodically update these assets, in which case it is the integrators responsibility to ensure they are using the latest version.

Step 3: Initiate payment request client-side using the Klarna payment button

Klarna requires the payment request be initiated using the Klarna payment button wherever possible. This allows for the simplest and best maintained client-side integration. If this is not possible, an alternate flow is described in the Klarna payment button cannot be initiated.

3.1. Add a `<div>` to your page where the Klarna payment button will be rendered:

Unset
```
<div id="#button-container"></div>
```

3.2. Use the klarna.js public endpoint in the Payment package:

JavaScript
```
var klarnaPaymentButton = klarna.Payment.button({
  id: "klarna-payment-button",
  shape: "pill",
  label: "Pay",
  theme: "dark"
});

klarnaPaymentButton.mount('#button-container');
```

3.3. Prepare the button click event configuration

Configure the InteractionMode

The payment request options allows you to specify the `interactionMode` parameter which controls how the Klarna purchase flow is launched (redirect / modal window) on different devices (mobile/desktop/native):

interactionMode	Definition
DEVICE_BEST	This is the **default value and recommended**. Klarna automatically selects the best flow depending on the device: ● Mobile: REDIRECT ● Desktop: Modal window if possible, fallback to REDIRECT. ● Native webview (mobile app) - REDIRECT **Note:** a `config.redirectUrl` is required in the payment request for this interaction mode

interactionMode	Definition
ON_PAGE	This value should be used only if redirection is not possible. The transaction happens on the same page by using a modal window if possible, if not then it will fallback to fullscreen iframe.
REDIRECT	Only redirect flow. **Note:** a `config.redirectUrl` is required in the payment request for this interaction mode

Generate a Redirect URL

The **`redirectUrl`** directs the shopper back to the merchant's website after a successful or aborted payment request. By incorporating placeholders into the URL, Klarna can dynamically insert relevant transaction information, ensuring the URL contains all necessary details for processing of a transaction.

Redirect URL example:

```javascript
config: {
  redirectUrl:
"https://partner.example/klarna-redirect?confirmation_token={klarna.payment_request.payment_confirmation_token}&request_id={klarna.payment_request.id}&state={klarna.payment_request.state}&reference={klarna.payment_request.payment_reference}"
}
```

These placeholders ensure that the redirect URL dynamically includes all necessary transaction details. Klarna will replace these placeholders with actual values before redirection, allowing partners to seamlessly handle the redirection process and ensure that all essential information is available for transaction processing. Details of the available placeholders are available below:

Placeholder	Description	Example
`{klarna.payment_request.payment_confirmation_token}`	The confirmation token, available when a transaction is successfully completed, and used to confirm the transaction.	krn:payment:eu1:confirmation-token:51bbf3db-940e-5cb3-9003-381f0cd731b7
`{klarna.payment_request.id}`	Klarna Payment Request identifier. Used in management of a Payment Request.	bebeabea-5651-67a4-a843-106cc3c9616a
`{klarna.payment_request.state}`	State of the payment request - may be used as a hint.	AUTHORIZED

`{klarna.payment_request.payment_reference}`	The provided reference to the payment request.	partner-payment-reference-12345

Payment request updates

A payment request can either be created explicitly by you, as integrator, using the Klarna payment request or implicitly via the Klarna payment button Interface:

- Once the Web SDK is loaded and a payment request is created, that same it is used throughout the entire session.
- The payment request cannot be retrieved server-side until it is not shared with Klarna when the `initiate()` method is called.
- When using the Klarna payment button, the initial payment request is passed in the click handler. The click handler allows you to modify the payment request data and choose if you want to initiate the payment.

Both the `request()` and `initiate()` methods share the same set of input parameters.

- `request(paymentRequestData?, options?): Request`
- `initiate(paymentRequestData?, options?): Promise<void>`

The payment request data allows the integrator to provide properties for the ongoing purchase.

Parameter	Definition
`paymentAmount`	Total payment amount
`currency`	3-letter ISO 4217 currency code
`config (optional)`	Configuration object for the payment request, in which the redirectUrl? properties can be provided.
`config.paymentOptionId`	The identifier returned by Klarna's messaging API or Web SDK to support the preselection of payment options within the Klarna Payments flow. This identifier is required if Klarna is presented as more than one payment option.
`config.shoppingSessionId`	The identifier returned by Klarna's Web SDK which ties a customer's activities together across multiple services. If Klarna's Web SDK is integrated by the partner, this identifier must be mapped to the shoppingSessionId returned by Klarna. Acquiring Partners must make this value available to merchants, and allow merchants to pass this through to Klarna at all interactions.
`merchantReference (optional)`	Used for storing the shopper-facing transaction number. It will be displayed to shoppers on the Klarna app and other communications. It will also be included in settlement reports for the purpose of reconciliation.

Parameter	Definition
`paymentReference (optional)`	For Distribution Partners: Reference to the payment request which can be used by distribution partners for the purpose of correlating your payment resource with the Klarna Payment Request.
`customer (optional)`	This represents who the customer is according to the merchant. These data points may be used by Klarna to simplify sign-up and during fraud assessment, they will not be used for underwriting and will not be persisted on created Payment Authorizations
`shipping (optional)`	Shipping information for the purchase. This data is used for fraud detection and shopper communication. If the purchase contains multiple shipments with different recipients, you must provide one shipping object per shipment.
`lineItems (optional)`	Line items describing the purchase, the total sum of the line items must match the payment amount.

Consult the **SDK reference** for a complete description of the request body parameters

3.4. Handle the button click event.

When using the Klarna payment button, the initial payment request is passed in the click handler, where you can modify the request and initiate it. Configurations available for the request are detailed below.

```javascript
JavaScript
klarnaPaymentButton.on("click", async (paymentRequest) => {
  paymentRequest.initiate(
    {
      paymentAmount: 9999,
      currency: "EUR",
      config: {
        redirectUrl:
"https://example.com?id={klarna.payment_request.id}andtoken={klarna.payment_request.paymen
t_confirmation_token}"
      }
    },
    {
      interactionMode: "DEVICE_BEST"
    });
});
```

Upon being redirected, the shopper will enter Klarna's payment flow. This flow allows the shopper to choose their payment method and input any necessary information. The process also may require the shopper to log in to their Klarna account to proceed with the payment.

⚠️ Be mindful when sharing shopper data:

- The merchant is required to, while respecting the privacy of their shoppers, send all applicable shopper data points when initiating the payment request which is then prefilled in the funnel.
- The merchant is responsible for disclosing how and when personal information is shared with their partners
- Returning shoppers checking out from a device where they are logged-in would skip the authentication step in the funnel.

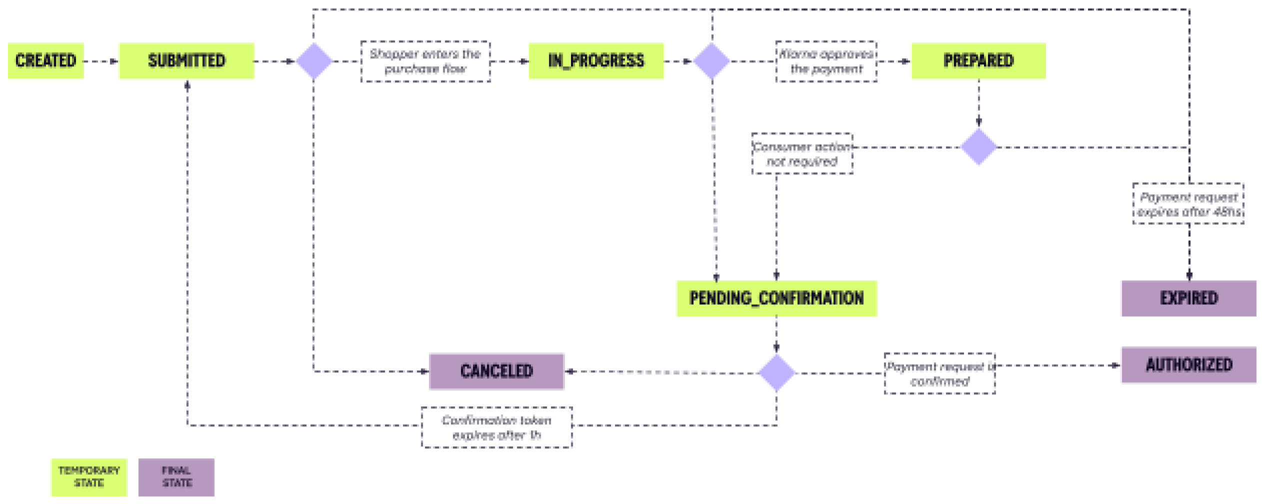
Step 4: Monitor payment state and retrieve payment confirmation token

During the checkout process, the payment request will transition to various states. Find below an overview of the possible transaction states together with a transition diagram:

Payment request state

State	Definition
`CREATED`	Payment request has been created client side, can freely be modified (Not valid for server side integration)
`SUBMITTED`	Payment request has been submitted to the backend and ready to be initiated or prepared. Request can be modified but must be synchronized
`IN_PROGRESS`	The payment flow is in progress (shopper is inside the purchase flow)
`PREPARED`	The payment request is prepared, but not yet finalized. This state can be triggered when using the `prepare()` method in the context of a multi-step checkout It must be finalized by calling `initiate()`
`PENDING_CONFIRMATION`	The payment flow has successfully been completed by the shopper and is pending final confirmation to complete the request.
`EXPIRED` *(final)*	The payment request has expired(t≥48h).
`AUTHORIZED` *(final)*	The payment request is authorized and a transaction has been created.
`CANCELED` *(final)*	The payment request has been canceled by the integrator. The `payment_confirmation_token` can only be CANCELED until the request is AUTHORIZED. After authorization, the cancellation is no longer possible.

Payment request state diagram


CREATED
SUBMITTED
Shopper enters the purchase flow
IN_PROGRESS
Klarna approves the payment
PREPARED
Consumer action not required
Payment request expires after 48hs
PENDING_CONFIRMATION
EXPIRED
CANCELED
Payment request is confirmed
AUTHORIZED
Confirmation token expires after 1h
TEMPORARY STATE
FINAL STATE

Once the shopper has successfully completed the payment on the purchase flow, the payment request will reach the state PENDING_CONFIRMATION and a `payment_confirmation_token` will be returned by Klarna. This is a unique identifier which is necessary to securely perform a server-side confirmation of the payment request.

There are several ways to monitor the payment state and retrieve the `payment_confirmation_token`. It is important to ensure you handle errors and fail gracefully regardless of the outcome of the transaction, see here for more information on integration resilience.

1. Subscribing to webhook events

To configure your webhooks please follow the guidelines on this section.

The webhook triggered when the payment request reaches the state PENDING_CONFIRMATION will contain the `payment_confirmation_token`.

Request example:

```
Unset
{
  "metadata": {
       "event_id": "e911ddab-f2c9-4d4c-aa2e-1954b290a91a",
       "event_type": "payment.request.state-change.pending-confirmation",
       "event_version": "v1",
       "occurred_at": "2024-05-27T14:41:30Z",
       "account_id":
"krn:partner:global:account:test:d6312901-1056-4231-8adb-d5abea7f3f8c",
       "product_instance_id":
"krn:partner:global:payment-product:test:54f2ac72-2839-4004-9560-a8dcd128868c",
       "webhook_id":
"krn:partner:global:notification:webhook:96420b72-8c4c-4554-9b97-dcb67272d513",
       "live": false
  },
  "payload": {
       "payment_request_id":
"krn:payment:eu1:request:efb8cf04-07f8-6dad-a49a-c6b6048b25e3",
       "payment_reference": "09e7a0f1-4207-4abe-b472-64559b14cc80",
       "merchant_reference": "e60fd9e1-f1a3-4762-9555-e9aa5169cec5",
       "state": "PENDING_CONFIRMATION",
       "previous_state": "IN_PROGRESS",
       "state_expires_at": "2024-05-27T15:41:30Z",
       "expires_at": "2024-05-29T14:40:58Z",
       "created_at": "2024-05-27T14:40:58Z",
       "updated_at": "2024-05-27T14:41:30Z",
       "payment_confirmation_token":
"krn:payment:eu1:confirmation-token:2db97a21-6706-6f0e-89ac-faf493be15ae"
  }
}
```

The content of the `payload{}` will slightly differ depending on the payment request state.

Consult the [API reference] for a complete description of the body parameters.

2. Client-side payment request state updates

The `on(event, callback): void` method in the Klarna SDK is designed to help you to efficiently manage payment request state transitions. By registering an event handler, you can respond dynamically to updates related to the ongoing payment request. It is important to note that the event handler may be triggered even if there is no state transition.

When involved in a redirection flow, the update handler activates once your page has loaded. This feature ensures that all pending updates are handled immediately upon registration of the event handler, eliminating the need for polling. This is particularly useful for maintaining smooth and responsive payment processing workflows.

Additionally, the `PaymentRequest` provides access to various properties of the ongoing payment request. This access allows developers to handle payment requests with greater precision and awareness of the transaction's current state:

Parameter	Type	Definition
`paymentRequestId`	String	Unique identifier of this payment request
`paymentRequest`	Object	The context that was authorized by this payment request. This is always identical to the input given by the merchant (`paymentRequestData`)

`state`	Enum	Current state of the payment request
`stateContext`	Object	State specific context for the ongoing state. The `paymentConfirmationToken` will be stored in this object once the payment request reaches the state `PENDING_CONFIRMATION`

To implement this method, follow these steps:

- Define your callback function that will handle specific events related to payment requests.
- Register this callback function using the `on(event, callback)` method.

Example:

```
JavaScript
klarna.Payment.on('update', (paymentRequest) => {
  console.log('Payment request state updated: ', paymentRequest.state)
})
```

3. Cancel the payment request

A payment request is open for 48h by default. Klarna recommends you to proactively close payment requests which did not result in successful transactions or where your session timeout is less than 48h. To match the default timeout window of your checkout you can trigger a DELETE request to `/v1/accounts/{account_id}/payment/requests/{payment_request_id}`.

The `paymentRequestId` of the ongoing payment request would first need to be retrieved client-side and sent to the back-end where the cancellation can be triggered.

Response example (State Canceled):

```
Unset
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "CANCELED",
  "previous_state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {},
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z",
  "payment_request": {
    "currency": "USD",
    "payment_amount": 1000,
    "config": {
      "redirect_url":
"https://partner.example/klarna-redirect?id={klarna.payment_request.id}"
    }
  }
}
```

4. Handling the redirect URL

If placeholders were implemented in the redirect URL as described in section 3.3, it's important to handle the details returned to ensure that the transaction information is processed correctly on your end. Once the shopper is redirected back to your website, the URL will contain the actual values replacing the placeholders. For example:

```
Unset
https://partner.example/klarna-redirect?confirmation_token=krn:payment:eu1:conf
irmation-token:51bbf3db-940e-5cb3-9003-381f0cd731b7&request_id=bebeabea-5651-67
a4-a843-106cc3c9616a&state=AUTHORIZED&reference=partner-payment-reference-12345
```

Parse the URL parameters to extract the transaction details, and pass these details to your systems accordingly. Validate that these parameters match your expectations, and the other methods through which this information is communicated. If desired, move to Step 5: Confirm Payment request server side.

Consult the **API reference** for a complete description of the request body parameters.

Step 5: Confirm Payment request server side

To confirm a payment request, you must provide the **payment confirmation token** that is returned after a successful payment flow.

⚠️ The payment confirmation token is valid for 60 minutes. You must confirm the payment within the time limit otherwise the transaction will be lost.

Ensure the payment is confirmed and the transaction authorized through a POST request to `/v1/accounts/{account_id}/payment/confirmation-tokens/{payment_confirmation_token}/confirm`.

Mandatory Request Parameter	Definition
`currency`	The currency in which the transaction is made.
`payment_amount`	The total amount to be charged, matching the sum of all line item amounts if any are provided.

Additional parameters can be submitted in the request. Consult the API reference for a complete description of the request body parameters.

This endpoint is **idempotent**, meaning the same confirmation request can be called multiple times with the same confirmation token, and it will return the same response each time. This ensures that the payment confirmation process is reliable and repeatable without any risk of double processing. More information on implementing Idempotency available here.

Request example:

```
Unset
{
    "currency": "EUR",
    "payment_amount": 1000
}
```

When the payment request has been successfully confirmed, Klarna will return an HTTP 200 Ok response and the response body will contain the `payment_transaction_id` which you will need to perform all payment transaction management.

Response example:

```
Unset
{
    "state_context": {
        "payment_transaction_id": "6c583364-786d-46c6-b5a7-0c8c88884038",
        "payment_pricing": {
            "rate": {
                "fixed": 0,
                "variable": 1700000
            }
    },
    "payment_request": {
        "currency": "SEK",
        "payment_amount": 1000,
        "config": {

       "redirect_url":"https://partner.example/klarna-redirect?id={klarna.payment_request
       .id}"
        }
    },
    "previous_state": "PENDING_CONFIRMATION",
    "payment_request_id": "krn:payment:eu1:request:bebeabea-5651-67a4-a843-106cc3c9616a",
    "state": "AUTHORIZED",
    "state_expires_at": "2024-05-29T09:37:36.849370204Z",
    "expires_at": "2024-05-29T09:37:36.849370204Z",
    "created_at": "2024-05-27T09:37:36.849370204Z",
    "updated_at": "2024-05-27T09:37:36.849370204Z"
    }
```

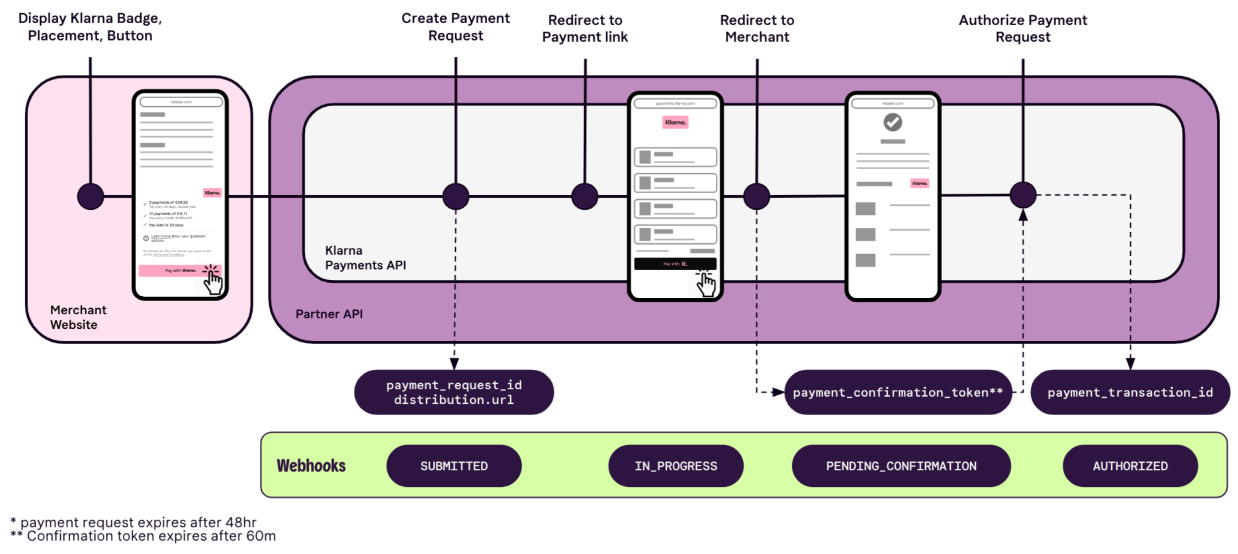
Consult the API reference for a complete description of the response body.

Alternative integration: Payment request initiated server-side

Overview

Use our server-side integration via REST API when not owning frontend assets in the payment flow.

Below is an illustration of the integration flow:



Step 1: Display Klarna payment badge, descriptor, and checkout placement

With this alternative integration path, the payment method selector is owned by the Merchants which are redirecting the shoppers to a payment URL provided by the PSPs.

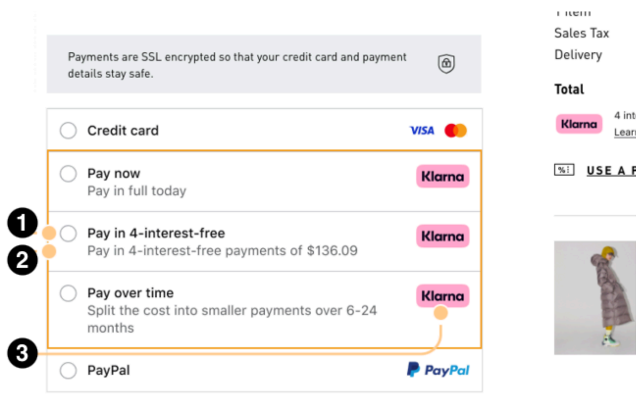
In a context of a Klarna integration, Merchants will have to build their payment method selector directly with the assets provided by Klarna

1.1: Get content for Klarna's payment badge, descriptor, and subheaders

1 Payment descriptor
High-level call to action. Provided in Klarna response.

2 Payment subheader
More detailed breakdown of the value of the option presented. Provided in Klarna response.

3 Klarna badge
Klarna logo. Provided in Klarna response.


Payments are SSL encrypted so that your credit card and payment details stay safe.
Credit card
VISA
Pay now
Pay in full today
Klarna
1
2
Pay in 4-interest-free
Pay in 4-interest-free payments of $136.09
Klarna
Pay over time
Split the cost into smaller payments over 6-24 months
Klarna
3
PayPal
PayPal
Sales Tax
Delivery
Total
Klarna
USE A

By using the `getPlacementContent()` method of klarna.js' `Messaging` package it is possible to retrieve information about descriptors, subheaders and the Klarna badge.

Release notes

📅 The `getPlacementContent()` method will be made available in a future release.

To achieve a flexible and global solution that allows merchants to choose their preferred presentation method, as per Enhance the shopping experience, we require you to **allow your merchants to pass the `message_preference` parameter at checkout**. If no preference is provided by your merchant, the default behavior is to display all available options separately.

Additionally, allow the passing of the `shopper_session_id` at this point to tie the session to any existing client-side sessions which may have been initiated by the merchant.

In response to the `getPlacementContext` method, you'll receive a `payment_option_id`. This must be returned to Klarna in the creation of a payment request unless Klarna is either being:

- Presented as a single payment option (using message_preference=Klarna) *or*
- Presented as a static payment method

If this is not done, the selection made within the merchant checkout will not be honored by Klarna and the customer may have to re-select their chosen payment method.

Parameters

Parameter	Definition
`locale`	Locale to use for returned content. BCP 47 (concatenation of language code (ISO 639-1 format) + "-" + country code (ISO 3166-1 alpha-2 format))

Parameter	Definition
	Example: en-US
`currency (optional)`	The purchase currency of the transaction. Formatted according to ISO 4217 standard, e.g. USD, EUR, SEK, GBP, etc. If not provided, default currency for the locale is used.
`payment_amount`	Total amount of a one-off purchase, including tax and any available discounts. The value should be in non-negative minor units. Eg: 25 Dollars should be 2500.
`message_preference (optional)`	Indicates the preferred payment option. Enum: ● PAY_NOW ● PAY_LATER ● PAY_OVER_TIME ● KLARNA

Example

Possible scenarios for `message_preference` parameter:

- If no `message_preference` is specified, the messaging copy for the **all-option approach** will be returned by default (default and recommended solution)
- If `message_preference`=PAY_LATER is specified, Klarna will return the messaging copy for the **Two-option approach** with Pay later as the promoted payment option. This will also apply to the other payment options.
- If `message_preference`=KLARNA is specified, Klarna will return the messaging copy for the **Single-option approach**

Find more information about available approaches here.

Request example:

```javascript
const descriptor = klarna.Messaging.getPlacementContent({
    key: "payment-descriptors",
    payment_amount: 20000,
    locale: "fr-CA"
})
```

In the response, Klarna will return an array of payment descriptors listing the content to be displayed (`PAYMENT_DESCRIPTOR`, `PAYMENT_DESCRIPTOR_SUBHEADER`, `KLARNA_BADGE`), indexed per `payment_option_id`.

The paymentOptionId is a parameter that should be used when initiating the payment request, as it allows you, as the acquiring partner, to preselect the corresponding payment option when the shopper enters the purchase flow.

Release notes

📅 PAYMENT_DESCRIPTOR_SUBHEADER will be added in a later release.

Response example:

```
Unset
{
  "paymentDescriptors": [
    {
      "paymentOptionId": "123xyz",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Pay now"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Pay in full today"
          },
          {
            "name": "KLARNA_BADGE",
            "type": "IMAGE",
            "url": "...",
            "label": "Klarna logo"
          }
        ]
      }
    },
    {
      "paymentOptionId": "123xyz",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Pay in 4-interest-free"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Pay in 4-interest-free payments of $25.00"
          },
          {
            "name": "KLARNA_BADGE",
                "type": "IMAGE",
                "url": "...",
                "label": "Klarna logo"
              }
            ]
          }
        },
        {
          "paymentOptionId": "123xyz",
          "content": {
            "nodes": [
              {
                "name": "PAYMENT_DESCRIPTOR",
                "type": "TEXT",
                "value": "Pay over time"
              },
              {
                "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
                "type": "TEXT",
                "value": "Split the cost into smaller payments over 6-24 months"
              },
              {
                "name": "KLARNA_BADGE",
                "type": "IMAGE",
                "url": "...",
                "label": "Klarna logo"
              }
            ]
          }
        }
      ]
    }
```

Option 2: Use the Partner Product Messaging API

Use the GET /v1/accounts/{account_id}/payment/messaging/payment-descriptors to retrieve the payment descriptors from Klarna.

`payment_amount` and `locale` are mandatory query parameters for this api call. `currency` and `message_preference` are optional query parameters and the logic here is the same as the one explained for the `getPlacementContent()`

Query Parameter	Definition
`locale`	Locale to use for returned content. BCP 47 (concatenation of language code (ISO 639-1 format) + "-" + country code (ISO 3166-1 alpha-2 format))

Query Parameter	Definition
	Example: en-US
`currency (optional)`	The purchase currency of the transaction. Formatted according to ISO 4217 standard, e.g. USD, EUR, SEK, GBP, etc. If not provided, default currency for the locale is used.
`payment_amount`	Total amount of a one-off purchase, including tax and any available discounts. The value should be in non-negative minor units. Eg: 25 Dollars should be 2500.
`message_preference (optional)`	Indicates the preferred payment option. Enum: ● PAY_NOW ● PAY_LATER ● PAY_OVER_TIME ● KLARNA

Example

Possible scenarios for `message_preference` parameter:

- If no `message_preference` is specified, the messaging copy for the **all-option approach** will be returned by default (default and recommended solution)
- If `message_preference`=PAY_LATER is specified, Klarna will return the messaging copy for the **Two-option approach** with Pay later as the promoted payment option. This will also apply to the other payment options.
- If `message_preference`=KLARNA is specified, Klarna will return the messaging copy for the **Single-option approach**

Check all approaches here.

In the response, Klarna will return an array of payment descriptors listing the content to be displayed (`PAYMENT_DESCRIPTOR, PAYMENT_DESCRIPTOR_SUBHEADER, KLARNA_BADGE`), indexed per `payment_option_id.`

The `payment_option_id` is a parameter that should be used when initiating the payment request, as it allows you, as the acquiring partner, to preselect the corresponding payment option when the shopper enters the purchase flow.

```
Unset
{
  "payment_descriptors": [
    {
      "payment_option_id": "edwqqr32dsad2dsefrassa",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "Financing"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Spread the cost over smaller montly installments"
          },
          {
            "name": "KLARNA_BADGE",
            "alt": "Klarna",
            "url": "https://osm.klarnaservices.com/images/logo_black_v2.svg",
            "type": "IMAGE"
          }
        ]
      }
    },
    {
      "payment_option_id": "czxsa4324132dsaddsxcxzzxs",
      "content": {
        "nodes": [
          {
            "name": "PAYMENT_DESCRIPTOR",
            "type": "TEXT",
            "value": "All options with Klarna"
          },
          {
            "name": "PAYMENT_DESCRIPTOR_SUBHEADER",
            "type": "TEXT",
            "value": "Pay now, in 30 days or with Financing"
          },
          {
            "name": "KLARNA_BADGE",
            "alt": "Klarna",
            "url": "https://osm.klarnaservices.com/images/logo_black_v2.svg",
            "type": "IMAGE"
          }
        ]
      }
    }
  ]
}
```

Option 3: Static assets

If relying on external dependencies is not feasible when building the checkout, or if you, the acquiring partner, do not own the client-side presentation of Klarna and cannot accept the `payment_option_id`, all assets (such as payment descriptors, payment subheaders, and the Klarna badge) are accessible as static content on docs.klarna.com. Klarna requires that up-to-date branding be in place in all checkouts, and it is the responsibility of the integrator to ensure that static assets remain current.

When using static assets, only the "one option" payment presentation will be available. Attempting to present multiple options will add friction, as any preselection made in your checkout will not persist within the Klarna session.

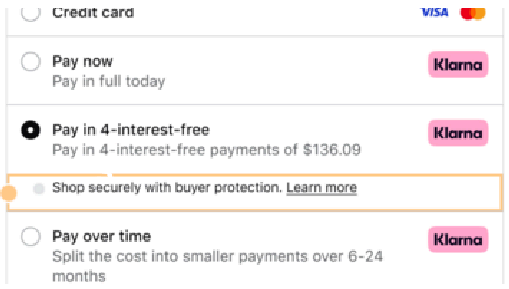
⚠️ Klarna may periodically update these assets, in which case it is the integrators responsibility to ensure they are using the latest version.

1.2: Display the Klarna checkout placement

The messaging package contains a custom element that can resolve into a collection of what we denote "placements''. These placements represent a form of visual text or imagery that represents the Klarna brand.

Checkout Placement

Displayed when the customer selects a Klarna payment option, this section includes a buyer protection USP and a "learn more" link.


Credit card

Pay now
Pay in full today

Pay in 4-interest-free
Pay in 4-interest-free payments of $136.09

Shop securely with buyer protection. Learn more

Pay over time
Split the cost into smaller payments over 6-24 months

There are three ways to display Klarna placement at the checkout:

Option 1: Use the SDK public endpoint in the messaging package:

```
JavaScript
var klarnaMessaging = klarna.Messaging.placement({
  key: "checkout",
  amount: 10000,
  currency: "USD",
  locale: "en-US"
}).mount("#checkout");
```

Option 2: Use the Partner Product Messaging API

Use the GET /v1/accounts/{account_id}/payment/messaging/checkout and specify a `locale` and `payment_amount` as query parameter.

The response will contain the necessary element for Partners to create the checkout placement themselves.

Unset
```
{
  "content": {
    "nodes": [
      {
        "name": "TEXT_MAIN",
        "type": "TEXT",
        "value": "Enjoy Buyer Protection with Klarna"
      },
      {
        "name": "ACTION_LEARN_MORE",
        "url": "url_t",
        "type": "ACTION",
        "label": "See payment options"
      },
      {
        "name": "ACTION_OPEN_BUYERS_PROTECTION_LINK",
        "url": "url_t",
        "type": "ACTION",
        "label": "Buyer Protection"
      },
      {
        "name": "KLARNA_BADGE",
        "alt": "Klarna",
        "url": "url_tg",
        "type": "IMAGE"
      }
    ]
  }
}
```

Option 3: Use the custom web component by invoking the following snippet:

Unset
```
<klarna-placement
  data-theme="dark"
  data-message-prefix="or"
  data-locale="en-GB"
  data-key="checkout"
  data-purchase-amount="24000.0"
  data-paymentOptionId="xxxxxx"
></klarna-placement>
```

Step 2: Create a payment request server-side

To start the purchase flow a payment request must be created. This request ensures that all necessary information is provided for a successful transaction. To start the process send a POST request to `/v1/accounts/{account_id}/payment/requests`.

The following data points are required for a successful request:

Parameter	Definition	Example
`currency`	The currency in which the transaction is made.	EUR
`payment_amount`	The total amount to be charged, matching the sum of all line item amounts if any are provided.	1000
`config.payment_option_id`	Specifies a payment option to be pre-selected in the purchase flow.	czxsa4324132dsaddsxcxzzxs
`config.shopping_session_token`	Unique identifier for a shopper session.	krn:shopping:eu1:session:...
`config.redirect_url`	The URL provided by the integrator where the shopper will be redirected after a successful purchase.	https://klarna.com?authorization_token={klarna.payment_request.payment_confirmation_token}and payment_request_state={klarna.payment_request.state}

Consult the API reference for a complete description of the request body parameters.

Request example:

```
Unset
{
    "currency": "EUR",
    "payment_amount": 1000,
    "config": {
        "payment_option_id":"czxsa4324132dsaddsxcxzzxs",
        "shopping_session_token": "krn:shopping:eu1:session:...",
         "redirect_url":
"https://partner.example/klarna-redirect?id={klarna.payment_request.id}"
    }
}
```

When the payment request has been successfully created, Klarna will return an **HTTP 201 Created**. The response body will contain the `payment_request_id` as well as the `state_context.distribution.url` to which the shopper should be redirected to.

Response example:

```
Unset
{
   "state_context": {
      "distribution": {
         "url":
"https://pay.test.klarna.com/eu/requests/b9bacf30-3619-60e5-bdcb-b0ad687d550b/start"
       }
     },
   "payment_request": {
       "currency": "EUR",
       "payment_amount": 1000,
       "config": {

       "redirect_url":"https://partner.example/klarna-redirect?id={klarna.payment_request
       .id}"
       }
    "payment_request_id": "krn:payment:eu1:request:b9bacf30-3619-60e5-bdcb-b0ad687d550b",
    "state": "SUBMITTED",
    "state_expires_at": "2024-05-29T08:33:10.088164687Z",
    "expires_at": "2024-05-29T08:33:10.088164687Z",
    "created_at": "2024-05-27T08:33:10.088164687Z",
    "updated_at": "2024-05-27T08:33:10.088164687Z"
}
```

Step 3: Redirect the shopper to Klarna purchase flow

After creating a payment request, the merchant should redirect the shopper to the start link provided in the response. This link is located at `state_context.distribution.url` in the response of the Step 2: Create a payment request server-side.

Shopping journey

Upon being redirected, the shopper will enter Klarna's payment flow. This flow allows the shopper to choose their payment method and input any necessary information. The process also requires the shopper to log in to their Klarna account, if needed, to proceed with the payment.

After completing these steps, the shopper is redirected to the URL specified in the `config.redirect_url`. This URL is provided in the initial payment request and ensures the shopper returns to the merchant's site after a successful transaction.

```
Unset
"config": {
   "redirect_url":
"https://merchant.com?authorization_token={klarna.payment_request.payment_confirmation_tok
en}"
}
```

Step 4: Monitor payment state and retrieve payment confirmation token

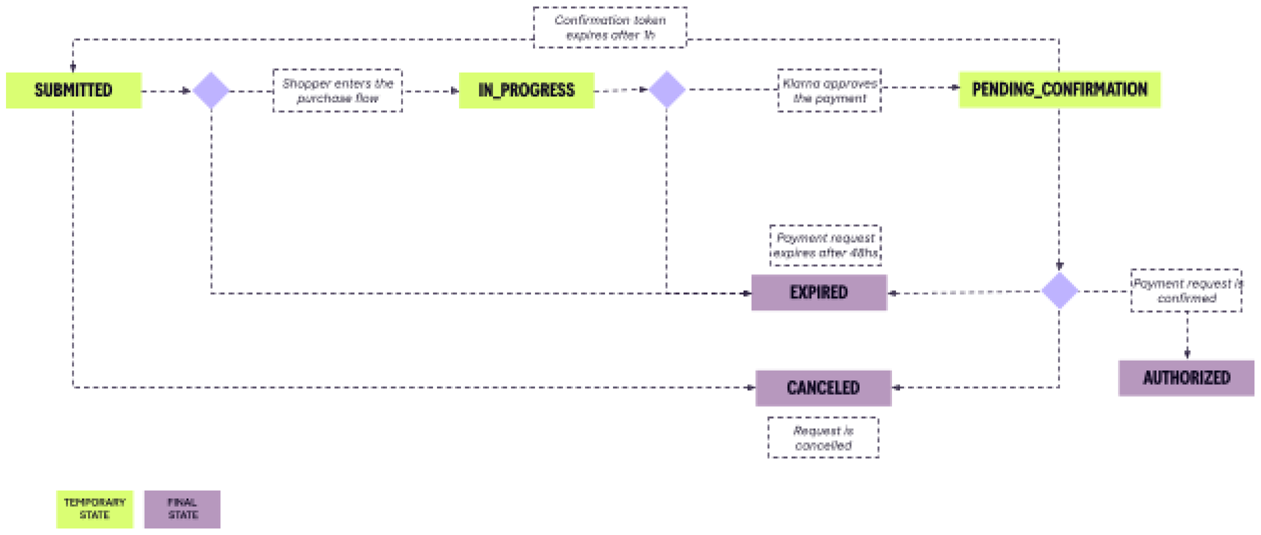
This section provides technical details on how to obtain the confirmation token from Klarna in the context of server-side only integrations. It also covers monitoring payment request states, relevant data elements, and properly closing the payment request.

The payment request will transition to various states during the Klarna payment flow, you will find below an overview of the possible payment request states together with a state transition diagram.

Payment request state

State	Definition
SUBMITTED	Payment request has been submitted to Klarna's backend and ready to be initiated, request can be modified but must be synchronized
IN_PROGRESS	The payment flow is in progress
PENDING_CONFIRMATION	The payment flow has successfully been completed by the shopper and is pending final confirmation to complete the payment request.
EXPIRED	The payment request has expired (t≥48h). This is a final state.
AUTHORIZED	The payment request is authorized - and a payment transaction has been created. This is a final state.
CANCELED	The payment request has been canceled by the integrator. This is a final state.

Payment request state diagram


Confirmation token expires after 1h
SUBMITTED
Shopper enters the purchase flow
IN_PROGRESS
Klarna approves the payment
PENDING_CONFIRMATION
Payment request expires after 48hs
EXPIRED
Payment request is confirmed
AUTHORIZED
CANCELED
Request is cancelled
TEMPORARY STATE
FINAL STATE

Once the shopper has successfully completed the payment in the purchase flow, the payment request will reach the state **`PENDING_CONFIRMATION`** and a **`payment_confirmation_token`** will be returned by Klarna. This is a unique identifier which is necessary to securely perform a server-side confirmation of the payment request.

There are several ways to monitor the payment state and retrieve the `payment_confirmation_token`. It is important to ensure you handle errors and fail gracefully regardless of the outcome of the transaction, for more information on integration resilience, see Integration resilience.

1. Subscribing to Webhook Events

To configure webhooks for your merchant accounts please follow the guidelines on this section.

The webhook triggered when the payment request reaches the state `PENDING_CONFIRMATION` will contain the `payment_confirmation_token`.

Example:

```
Unset
{
  "metadata": {
    "event_type": "payment.request.state-change.pending-confirmation",
    "event_id": "d9f9b1a0-5b1a-4b0e-9b0a-9e9b1a0d5b1a",
    "event_version": "v1",
    "occurred_at": "2024-01-01T12:00:00Z",
    "correlation_id": "2d1557e8-17c3-466c-924a-bbc3e91c2a02",
    "account_id": "krn:partner:account:206bbb83-9b6e-46fa-940d-337153c04a58",
    "product_instance_id":
"krn:partner:product:payment:ad71bc48-8a07-4919-a2c1-103dba3fc918",
    "webhook_id":
"krn:partner:global:notification:webhook:120e5b7e-abcd-4def-8a90-dca726e639b5",
    "live": true
  },
  "payload": {
    "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
    "payment_reference": "partner-payref-1234",
    "merchant_reference": "order-5678",
    "state": "PENDING_CONFIRMATION",
    "previous_state": "IN_PROGRESS",
    "payment_confirmation_token":
"krn:payment:eu1:confirmation-token:e15432a5-ebcc-45bc-934c-e61399db597b"
  }
}
```

The content of the `payload{}` will slightly differ depending on the payment request state.

Consult the **API reference** for a complete description of the webhook payload.

2. Placeholder in the redirect_url

The `redirect_url` directs the shopper back to the partners website to the predefined url after a successful or abandoned authorization. By incorporating placeholders into the URL, Klarna can dynamically insert relevant transaction information, ensuring the URL contains all necessary details for processing.

To ensure security and integrity, the server-side confirmation payment request call is required to complete an order. Klarna recommends all partners verify data received via redirect against that received via webhooks to prevent token misuse. Tokens passed via the redirect_url are only valid for the account that places the order, preventing hijacking.

Redirect URL example:

```JavaScript
config: {
  redirectUrl:
"https://partner.example/klarna-redirect?confirmation_token={klarna.payment_request.paymen
t_confirmation_token}andrequest_id={klarna.payment_request.id}andstate={klarna.payment_req
uest.state}andreference={klarna.payment_request.payment_reference}"
}
```

Where:

Placeholder	Example

Payment confirmation token	cd227fcd-21ee-4903-89ed-bd694144009e
payment request ID	bebeabea-5651-67a4-a843-106cc3c9616a
state	AUTHORIZED
payment reference	partner-payment-reference-12345

```
Unset
https://partner.example/klarna-redirect?confirmation_token=cd227fcd-21ee-4903-89ed-bd69414
4009eandrequest_id=bebeabea-5651-67a4-a843-106cc3c9616aandstate=AUTHORIZEDandreference=par
tner-payment-reference-12345
```

These placeholders ensure that the redirect URL dynamically includes all necessary transaction details. Klarna will replace these placeholders with actual values before redirection, allowing integrators to seamlessly handle the redirection process and ensure that all essential information is available for transaction processing.

```
Unset
{klarna.payment_request.id} - Klarna Payment Request identifier.
{klarna.payment_request.state} - State of the payment request - may be used as a hint.
{klarna.payment_request.payment_confirmation_token} - The confirmation token.
{klarna.payment_request.payment_reference} - Your reference to the payment request.
{klarna.payment_request.referer} - URL where the authorization started.
```

3. Read the Payment Request

Another way to obtain the confirmation token is by reading the payment request endpoint.

Using the read payment request endpoint to retrieve the confirmation token provides an alternative method for integrators to ensure they have the necessary token to complete the transaction. This method can be particularly useful for verifying the token or obtaining it after the initial redirection flow.

By leveraging both the redirect URL placeholders and the read payment request endpoint, merchants can ensure they have all the necessary tools to manage and confirm payments efficiently.

Fetch the token directly by sending a GET request to /v1/payment/{account_id}/requests/{payment_request_id}.

Response example (State Pending confirmation)

```
Unset
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "PENDING_CONFIRMATION",
  "previous_state": "PREPARED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "payment_confirmation_token":
"krn:payment:eu1:confirmation-token:e15432a5-ebcc-45bc-934c-e61399db597b"
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z",
  "payment_request": {
    "currency": "USD",
    "payment_amount": 1000,
    "config": {
      "redirect_url":
"https://partner.example/klarna-redirect?id={klarna.payment_request.id}"
    }
  }
}
```

Cancel the payment request

⚠️ A payment request is open for 48h, this period is not customizable. If you wish to match the length of a payment request, the payment request must be canceled via this request.

Klarna recommends you to proactively close payment requests which do not result in successful transactions and align this with the default timeout of your checkout process to avoid having shoppers going back to the Klarna Purchase Flow after your own checkout window has expired.

To match the default timeout window of your checkout you can trigger a DELETE request to `/v1/accounts/{account_id}/payment/requests/{payment_request_id}`.

Response example (State Canceled)

```
Unset
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "CANCELED",
  "previous_state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {},
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z",
  "payment_request": {
    "currency": "USD",
    "payment_amount": 1000,
    "config": {
      "redirect_url":
"https://partner.example/klarna-redirect?id={klarna.payment_request.id}"
    }
  }
}
```

Consult the **API reference** for a complete description of the request body parameters.

Step 5: Confirm Payment request server side

To confirm a payment request, you must provide the **payment confirmation token** that is returned after a successful payment flow.

⚠️ The payment confirmation token is valid for 60 minutes. You must confirm the payment within the time limit otherwise this would result in a lost transaction.

Ensure the payment is confirmed and the transaction authorized through a POST request to `/v1/accounts/{account_id}/confirmation-tokens/{payment_confirmation_token}/confirm`.

Mandatory Request Parameter	Definition
`currency`	The currency in which the transaction is made.
`payment_amount`	The total amount to be charged, matching the sum of all line item amounts if any are provided.

Additional parameters can be submitted in the request. Consult the API reference for a complete description of the request body parameters.

This endpoint is **idempotent**, the same confirmation request can be called multiple times with the same confirmation token, and it will return the same response each time. This ensures that the payment confirmation process is reliable and repeatable without any risk of double processing. More information on idempotency is available in Idempotency.

Request example:

```
Unset
{
    "currency": "EUR",
    "payment_amount": 1000
}
```

When the payment request has been successfully confirmed, Klarna will return an HTTP 200 Ok response and the response body will contain the **payment_transaction_id** which you will need to perform all payment transaction management.

Response example:

```
Unset
{
    "state_context": {
        "payment_transaction_id": "6c583364-786d-46c6-b5a7-0c8c88884038",
        "payment_pricing": {
            "rate": {
                "fixed": 0,
                "variable": 1700000
            }         }
    },
    "payment_request": {
        "currency": "SEK",
        "payment_amount": 1000,
        "config": {

       "redirect_url":"https://partner.example/klarna-redirect?id={klarna.payment_request
       .id}"
        }
    },
    "previous_state": "PENDING_CONFIRMATION",
    "payment_request_id": "krn:payment:eu1:request:bebeabea-5651-67a4-a843-106cc3c9616a",
    "state": "AUTHORIZED",
    "state_expires_at": "2024-05-29T09:37:36.849370204Z",
    "expires_at": "2024-05-29T09:37:36.849370204Z",
    "created_at": "2024-05-27T09:37:36.849370204Z",
    "updated_at": "2024-05-27T09:37:36.849370204Z"
}
```

Consult the **API reference** for a complete description of the response body.

More payment use cases

Klarna supports all possible payment scenarios for your merchants, accommodating any type of business, including those requiring subscriptions or on-demand solutions. Klarna's omnichannel solution gives your customers the same experience everywhere, online, in physical stores, and via mobile apps.

- Update value of a product in the transaction: Detail how to update the payment request to a SUBMITTED payment request to handle updates to an order.
- Subscriptions: enable subscription based use cases for your merchants.

- Mixed payments: enable baskets that include both types of payments one-time and recurrent/token creation.
- On-demand transactions: enable on-demand transactions and micropayments in your merchants payment flow.
- Mobile app: enable Klarna payments in mobile apps.
- In-store: enable Klarna payments in your merchants physical stores.
- Special segments:enable Klarna for specific business models such as travel or digital services such as gaming or others.

Updating an order

Server-side updates

Klarna allows you to update a payment request through a PATCH request to `/v1/accounts/{account_id}/payment/requests/{payment_request_id}`.

⚠️ This can only be performed when the payment request is in the **SUBMITTED state**.

This restriction ensures that the shopper is aware of any modifications to the payment request.

Consult the **API reference** for a complete description of the request body parameters.

Request example:

```javascript
{
   "payment_reference": "payment-reference-2024-05-27T12:50:42.677Z",
   "merchant_reference": "merchant-reference-e9c9b713-651d-4f26-bc9d-cc573f616134",
}
```

Response example:

```javascript
{
   "state_context": {
      "distribution": {
         "url":
"https://pay.test.klarna.com/eu/requests/b9bacf30-3619-60e5-bdcb-b0ad687d550b/start"
       }
     },
   "payment_reference": "payment-reference-2024-05-27T12:50:42.677Z",
   "merchant_reference": "merchant-reference-e9c9b713-651d-4f26-bc9d-cc573f616134",
   "payment_request": {
       "currency": "EUR",
       "payment_amount": 1000,
       "config": {
       "redirect_url":"https://partner.example/klarna-redirect?id={klarna.payment_request
       .id}"
       }
   "payment_request_id": "krn:payment:eu1:request:b9bacf30-3619-60e5-bdcb-b0ad687d550b",
   "state": "SUBMITTED",
   "state_expires_at": "2024-05-29T08:33:10.088164687Z",
   "expires_at": "2024-05-29T08:33:10.088164687Z",
   "created_at": "2024-05-27T08:33:10.088164687Z",
   "updated_at": "2024-05-27T08:33:10.088164687Z"
}
```

Client-side updates

Klarna.js allows updates to an ongoing payment request that has the SUBMITTED state through a variety of different methods.

Request
Passing payment request data to the `request()` method updates the local state of the payment request with the information passed.

JavaScript
```javascript
try {
  const paymentRequest = Klarna.Payment.request({
    currency: 'EUR',
    paymentAmount: 1000,
    config: {
      redirectUrl: 'https://example.com'
    },
  })
} catch (error) {
  // Handle errors
}
```

If a payment request was created on the server-side using the same account as the client-side, you can pass the existing payment_request_ID. If you provide new payment request data, it will overwrite the original data from the server-side creation. If no new data is provided, the original server-side data will be used.

JavaScript
```javascript
Klarna.Payment.request('krn:payment:eu1:request:b9bacf30-3619-60e5-bdcb-b0ad687d550b')
.initiate()
```

Update
Calling update on an existing payment request will sync the payment request state to the backend as well as update the local state.

```javascript
try {
  paymentRequest.update({
    currency: 'EUR',
    paymentAmount: 9000,
    config: {
      redirectUrl: 'https://example.com'
    },
  })
} catch (error) {
  // Handle errors
}
```

Initiate
Passing payment request data to the `initiate()` method is also possible before initiating the payment request for last minute modifications of the payment request such as shipping fees, tax adjustments, or final additions to the cart . If the `initiate()` method is called without any parameters, the local state of the payment request will be used to initiate the payment request.

```javascript
try {
  paymentRequest.initiate({
    currency: 'EUR',
    paymentAmount: 11000,
    config: {
      redirectUrl: 'https://example.com'
    },
  }, {
    interactionMode: 'DEVICE_BEST'
  })
} catch (error) {
  // Handle errors
}
```

Klarna payment button cannot be initiated

This section continues from Initiate a payment request using the Klarna payment button, and is meant to guide integrators through the necessary steps if the Klarna payment button is not an available integration type.

1. Add a button to your page:

```
JavaScript
<button type="button" id="BUY">Buy</button>
```

2. Create the payment request :

Details on how to configure this request are outlined in 3.3. Prepare the button click event configuration.

```
JavaScript
const paymentRequest = klarna.Payment.request(
  {
    paymentAmount: 9999,
    currency: "EUR",
    config: {
      redirectUrl:
"https://example.com?id={klarna.payment_request.id}andtoken={klarna.payment_request.paymen
t_confirmation_token}"
    }
  },
  {
    interactionMode: "DEVICE_BEST"
  }
);
```

3. Initiate payment request when shopper clicks buy button:

```
JavaScript
var initiate = document.getElementById("BUY");


initiate.addEventListener("click", function (event) {
  paymentRequest.initiate();
});
```

For details regarding next steps, continue with Step 4: Monitor payment state and retrieve payment confirmation token.

Consult the **payment flow** section for an overview of the shopper journey.

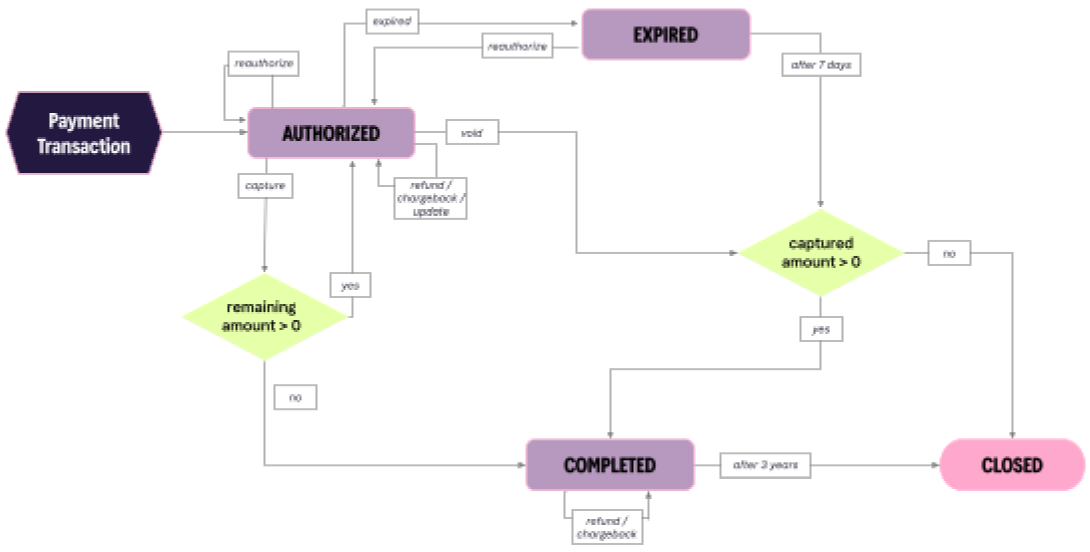
Manage Klarna payment transaction

The Payment transaction API enables you to manage and orchestrate the entire lifecycle of a payment, from the moment it's authorized to when it's utilized and beyond. This API allows you to track payments, update details, manage financial transactions through captures, refunds and generally keep everything running smoothly behind the scenes.

Payment transaction state definition

When managing payment transactions, it is crucial to understand the different states that a payment can transition through during its lifecycle. Each state represents a specific phase in the payment process, dictating what actions can be taken and what limitations are in place. Below, we provide definitions for some key states encountered within the payment transaction API.

- **Authorized**: Identifies a payment that has been authorized but is not yet fully captured. It is waiting for further actions such as capture, refund, update or void.
- **Expired**: Indicates that the payment transaction has reached its lifespan without being completed. A transaction expires if it is not fully captured within a set period of time, defined in the partner's contract. An expired payment transaction transitions to the `CLOSED` state if it is not reauthorized within 7 days from the expiry.
- **Closed**: Indicates that the authorization has reached its definitive conclusion, or end of life. In this state, no further operations, including refunds, can be executed. Following are the conditions for the transition:
    - **Post Completed**: A payment transaction in the Completed state reaches its end of life after 3 years.
    - **Post Expiry**: The payment transaction transitions to this state if it hasn't been completed and has surpassed the 7-day grace period following expiry.
- **Completed**: indicate a state where the payment transaction has been finalized through a funds transfer. In essence, a payment transaction is considered completed when any amount is captured and no more authorized amount remains.
    - **Fully captured**: The funds corresponding to the full authorization amount have been successfully captured.
    - **Partially captured and expired:** The payment transaction has been partially captured, and the 7-day grace period has passed upon expiration. Any remaining authorization is released, and the payment authorization is considered to be completed.



Read payment transaction

The API call is used to retrieve detailed information about a transaction through your system. When the need arises to check the specifics of a transaction, this API call empowers you with essential details such as the current state, authorized amount, currency, line items and shipping information. This ensures that you have real-time access to crucial transaction information, enabling efficient transaction management and customer service. The API's response allows you to stay updated on the transaction status and take any necessary actions. Here are the actions you can take for different states:

- Authorized: When a payment is in the AUTHORIZED state, the following actions can be taken: capture, refund, update, and void.
- Expired: When a payment is in the EXPIRED state, the options are more limited:
  - Reauthorize: If within the 7-day grace period, you can reauthorize the payment to extend its lifespan and allow for capture.
- Completed: When a payment is in the COMPLETED state, no further operations such as captures can be performed, but refunds can still be executed.
- Closed: When a payment is in the CLOSED state, no further operations such as captures, refunds, or voids can be performed. The transaction is considered finalized and cannot be reopened or modified.

Consult the **API reference** for a complete description of the request body parameters.

Update Payment Transaction

The update payment transaction allows you to update the details of a transaction, specifically focusing on the merchant references and tracking information.

- The shipment details may be updated to include delivery tracking immediately after purchase, during capture, or even after capturing it. This improves the customer experience and is strongly encouraged.

- If you require custom transaction IDs or need to adjust the default IDs provided by Klarna to suit your internal processes, customizing the merchant reference is necessary. The merchant_reference is used for storing the customer-facing transaction number and is crucial for transaction tracking. This identifier will be displayed to customers on the Klarna app and other communications.

Consult the **API reference** for a complete description of the request body parameters.

Capture Payment Transaction

The capture payment transaction API is meant to be used once a transaction has been fulfilled, which signifies that the goods have been shipped to the customer. This API allows you to capture either the entire payment amount or a partial amount, based on your specific needs.

- Ensure that the capture amount does not exceed the payment amount unless you have reauthorized your payment transaction to a higher payment amount.
- Within the capture call, you can include changes to the line items. This is useful if a different good with the same amount is shipped.
- Including the shipping info object is recommended for better tracking and record-keeping.

Consult the **API reference** for a complete description of the request body parameters.

Void payment transaction

The Void payment transaction call is used when customers opt to cancel a payment transaction. This can be done before the transaction is fully captured. The authorized amount that has not been captured will be released, and no further updates to the transaction will be allowed. This API call can also be used to release the remaining authorization of an order and to signal that there is no intention to perform further captures. When implementing the void call, please follow these guidelines:

- Partial Captures: If the transaction already has some captures, only the uncaptured amount will be voided.
- No Captures: If the transaction doesn't have any captures, the entire amount will be voided.

Consult the **API reference** for a complete description of the request body parameters.

Refund payment capture

The refund capture call is utilized to initiate the refund process when customers return items they've purchased. Note that this call is associated with a specific payment capture.

If in any situation you will have multiple captures on a transaction (for example split shipments),you are required to implement this path.

Consult the **API reference** for a complete description of the request body parameters.

Refund payment transaction

The refund transaction call is utilized to initiate the refund process against any arbitrary amount, up to the total amount captured on an overall transaction.

⚠️ If you are supporting scenarios involving multiple captures you must use the Refund payment capture endpoint to refund captures individually.

Consult the **API reference** for a complete description of the request body parameters.

Re authorize a payment transaction

This operation allows you to obtain a new authorization with a new expiry date, providing additional time to capture and fulfill it. It can only be performed

- Before the payment transaction expires.
- Within 7 days after the expiry, you should aim to reauthorize before that time and before the transaction is in a closed state.

During reauthorization, the payment amount, line items, and shipping address can be changed. A risk assessment will be conducted for this request.

Release notes

📅 Re-authorize feature will be made available in a future release.

Pricing and reconciliation

How pricing works

When a transaction is confirmed, Klarna issues rate details based on the transaction specifics in a "promise". This rate confirmation appears in the `confirm_payment` request API response and is provisional. The application of these fees depends on the number of captures and the captured amount, as both variable and fixed fees are applied per capture.

The rate provided should not be shared with merchants as a final amount, as it is subject to calculation based on captured amount and quantity, and is reflective of your `buy_rates` as an acquiring partner.

Should a “microtransaction cap” be applied to the transaction, the rate charged on the transaction will have a maximum applied, based on the total value of the payment. In this case the rate will reflect this maximum amount. A microtransaction is defined within the pricing framework, and may vary by country or MCC.

We advise you to use the rates provided in the confirm payment request response for calculations and refrain from storing these rates. Should retrieval of these rates be necessary, you can access them using the GET request `/v1/accounts/`**`{account_id}`**`payment/requests/{payment_request_id}`. To view the broader rates across their pricing plan, you should utilize the GET read a price plan request. This ensures accurate and up-to-date rate application in all transactions and calculations.

Release notes

📅 GET read a price plan request *will be supported in future releases.*

Definitions and Calculations

- **Effective rates**: These rates are linked to a specific price plan identified by a `price_plan_id`. Rate details for a price plan can be accessed through the Partner management API using this ID. Effective rates consider factors like the purchase country, MCC code, sales channel (in-store, online store), and payment program. The fixed and variable costs shared on confirmation are applied to a transaction on a per capture basis.

- **Discounts and penalties**: Discounts reduce transaction fees based on certain criteria or promotions. Penalties are extra charges for non-compliance with terms or atypical transaction patterns. The effective rate communicated accounts for any applied discounts or penalties.

- **Rate communication**: Within the confirm payment request response, rate details are disclosed. The rate object shows the amounts calculated based on these details:

Fee	Definition
Fixed	Minor units (currency defined by the transaction currency)
Variable	Percentage points representing percent value multiplied by 1 000 000. E.g: 1.7% is transmitted as 1700000

Response example:

```
Unset
"payment_pricing": {
        "rate": {
                "fixed": 100,
                "variable": 1200000
        }
}
```

Release notes

📅 Settlement API will be made available in a future release.

Reconciling Klarna Settlements

Once a transaction is captured, the effective rates are applied based on the capture amount. The detailed settlement file will include the actual calculated amount of the fees, total amount, net amount, and VAT applied to any fees.

Relating a payment to a settlement file

Payouts are made to the bank account(s) according to the schedule defined in your settlement configuration. The `settlement_id` is contained within the payment according to best practices in each market. This id be used to easily correlate a given payout to the associated settlement file. Each currency will result in a separate payout with its own settlement id.

Relating a transaction to the applied fees

A given transaction can be captured or refunded multiple times, and as such fees may be applied to the captured amount multiple times.

- The `payment_transaction_id` is provided within the settlement file for all actions regarding a payment transaction.

- The `payment_capture_id` reference is only associated with a specific capture, and `payment_refund_id` with a specific refund. These references may be useful in associating a given action with its reconciliation.
- The `dispute_id` used to handle disputes is also contained within the settlement file, and can be used to understand the associated fees applied as a result of that action.

Further details on the fields included within a settlement file are available in the Settlements section.

Handling adjustments

The settlement file will detail captures, fees, refunds, disputes, discounts, and penalties. Here's how to handle them:

- **Fees**: The settlement file will include the fees applied to each event. These match the effective rates as previously discussed. VAT will be applied to the fees if applicable. The fixed and variable rates of the VAT will be included in the settlement file.
- **Refunds**: Refunds are withheld from subsequent payouts, as the payment is NET. If the settlement amount is zero, the outstanding debt will be carried over and applied to subsequent settlements.
  - **Debt statement**: Debt statements are issued if the partner has a negative balance (at an aggregate level) for more than 30 days. This statement is not an invoice but a statement of the partner to pay Klarna of the negative balance.
- **Disputes**: Disputes are also withheld (if won by shopper), and a fee may be applied.
- **Discounts**: Discounts may be applied for the implementation of boost products or fulfilling other integration requirements as defined by Klarna. Discounts are applied on a per transaction basis.
- **Penalties**: Penalties may be applied for deviating from best practices or miscommunicating Klarna in some other way. Penalties may be applied on a per transaction basis.

Consuming rate and fee data

Reconciliation requires an understanding of both the buy rates communicated by Klarna and the sell rates applied to merchants.

- **Rate data**: The buy rates are included within the `price_plans` and can be checked at any time, but are defined in conversation with Klarna. Sell rates belong entirely to you as acquiring partner, but are generally expected to match Klarna's public rates.
- **Fee calculation**: Use this data to calculate the fees for each transaction. Transactional fees are determined through retrieval of the fees defined in the price plan. Any applicable discounts are subtracted, and relevant penalties are added. If a microtransaction cap applies to the order, the lesser of the calculated fee or the cap will be used. The variable rate is applied at the time of capture, and the final result will be detailed in the settlement reports.

Communicating costs to merchants

Ensure clear communication with merchants regarding fees and costs to maintain transparency and reduce risk of shopper errands and chargebacks. Provide detailed information about fixed and variable fees, adjusted to reflect the sell rates. If there is an issue, a merchant should raise the confusion with the partner, as Klarna may be unable to communicate costs with the merchant directly.

It's essential to manage the differences between the rates you buy from Klarna and the rates you sell to merchants. Perform internal calculations to ensure sell rates accurately reflect the buy rates plus any margin. Ensure the sell rates communicated to merchants are correct and transparent. Typically, the fixed and variable fees on a transaction should be communicated to the merchant after they are adjusted to reflect the sell rates of the partner.

Settlement File

Data contained within the detailed settlement file includes:

Parameter	Definition	Example
`captured_at`	Datetime when the event was registered in Klarna's system (Coordinated Universal Time, UTC). In case of a SALE transaction it refers to the moment when you shipped the goods to the shopper and captured/activated the payment.	2018-08-10T07:45:00Z
`created_at`	Date when the transaction was first created in Klarna's system (Coordinated Universal Time, UTC).	2018-08-10T07:45:00Z
`payment_transaction_created_at`	Date of the transaction confirmation (Coordinated Universal Time, UTC).	2018-08-10T07:45:00Z
`payment_transaction_id`	Unique identifier of the transaction. All related transactions in the life-span of a transaction are associated with this ID. eg. fees or refunds. It is therefore the recommended identifier for reconciling the report lines with your system.	c504a9bb-1948-46d5
`klarna_reference`	The reference to identify the shopper-facing transaction.	9875QW2
`payment_capture_id`	Unique identifier for every capture on a transaction and only provided for sale and fee transactions. In case of partial shipments a transaction is captured more than one time. Each related capture is reflected as a sale transaction with a unique capture_ID.	8e93b66-6ab1-4d3d-b60d-1cc4e24f4a99
`merchant_reference`	The internal reference to the transaction, submitted during transaction creation.	c504a9bb-1948-46d5
`payment_transaction_reference`	The internal reference to the transaction, submitted during payment creation. Not shared with end customers.	c504a9bb-1948-46d5
`amount`	The amount of the action represented by this settlement line in minor units, denominated by the currency.	$100.00 is 10000
`payment_pricing.details.price_plan_id`	ID assigned to the price plan defined for a acquiring partner	krn:partner:global:pricing:payments:price-plan:171080e7-2637-4675-a224-ec032723ebdf

Parameter	Definition	Example
`payment_pricing.rate.fixed`	Value of fixed fee in non-negative minor units	$5.99 is 599
`payment_pricing.rate.variable`	Percentage points representing percent value multiplied by 1 000 000	1.5% is 1 500 000
`payment_pricing.rate.tax.variable`	VAT (Value added tax in Europe) or GST (goods and services tax in Australia) rate on Klarna fees. Percentage points representing percent value multiplied by 1 000 000	1.2% for 1 200 000
`payment_pricing.rate.tax.fixed`	VAT (Value added tax in Europe) or GST (goods and services tax in Australia) amount on Klarna fees. Represented in minor units.	$1.20 is expressed as 120
`discount_amount`	Discounts applied to the capture by Klarna, must be relayed to the end merchant in a 1:1 ratio.	$1.20 is expressed as 120
`penalty_amount`	Penalties applied to the capture by Klarna, must be relayed to the end merchant in a 1:1 ratio.	$1.20 is expressed as 120
`currency`	Currency in which the payment has been registered. The following currencies are currently available:DKK, EUR, GBP, NOK, SEK, USD, CHF, CAD, AUD	EUR
`payment_refund_id`	Unique identifier for Return and Reversal transactions. In case of partial returns, each return transaction is associated with a unique refund_ID.	8e93b66-6ab1-4d3d-b60d-1cc4e24f4a99
`payment_option_category`	The payment option category, like PAY_NOW, PAY_LATER etc.	PAY_NOW
`capture_reference`	The reference to the capture, submitted during capturing an transaction via API.	43d2fd82-4c4b-412a-bd8b-07def0f1b721
`refund_reference`	The reference to the refund submitted during refunding a transaction via API.	7586ca7a-a92d-48ec-be62-628c30d8c615
`dispute_id`	Unique identifier issued by Klarna to identify a dispute, facilitating efficient tracking of its status.	krn:klarna:us1:dispute:return:318513301950489
`account_id`	Unique account identifier assigned by Klarna to the onboarded merchant	krn:partner:global:account:live:LWT2XJSE
`settlement_id`	Unique identifier for the settlement and payout, will be the reference on the bank statement	

Test your integration

Klarna's test mode allows you to test your integration without making actual charges or payments. Test mode is a testing environment to simulate transaction flows without moving actual money, to ensure that your integration with Klarna's systems operates as expected. Utilizing this mode is essential for identifying and resolving any issues and should be included in all release routines to ensure proper testing before going live.

Klarna requires all integrators to complete rigorous validation in the test environment prior to launching in the live environment, and to provide Klarna with comprehensive access to their testing environment for further validation.

Any testing within the live environment should be approached with the understanding that rejections are a natural part of Klarna validations and can occur as part of the live testing phase. Although live environment testing is not recommended, if it is necessary the below reasons are common contributing factors to rejections in the live environment:

- Inputting test data (i.e. anything other than your real name, personal email, personal mobile, home billing address, etc)
- Using a company address as personal data
- Insufficient purchase history with Klarna combined with high-value or large quantities of purchases
- Triggering velocity rules - Loading the checkout multiple times in a short space of time from the same device/IP

Test cases

In this section, you can find different scenarios for testing Klarna payments and management API flows.

Management API test cases

Test your Klarna Partner Management API integration by following the test cases below. As a Klarna acquiring partner, we require you to complete and pass all the test cases listed in this section unless exceptions have been approved by Klarna. If a specific product or interaction with Klarna is not included in your integration, and this has been documented in your Solution Scope Document, the representative test case should be skipped.

☑Test case 1: Create and list credentials for your own account

Steps to follow:

1. Create new API credentials for your own Acquiring Partner account.
2. List the credentials to inspect all created API credentials.

Security Warning

⚠️ *For security reasons never provide real personal or business data in a test environment.*

☑Test case 2: Onboard a new merchant

Steps to follow:

1. Onboard a new merchant and get the credentials.
2. Validate all the account and business info, channel collections, and all the other important details are sent to Klarna.

You can use this sample data found on docs.klarna.com.

☑Test case 3: Disable a payment product and revert the action

Acquiring Partners working with Klarna can proactively suspend a product associated with an account if they stop their relationship with that account holder or believe the account may be breaking the terms of their agreement.

Steps to follow:

1. Onboard a new merchant and get the credentials.
2. Suspend a payment product.
3. Revert the suspension.

☑Test case 4: Configure, update and list channels for an account

Channels represent how Klarna's products are shown to the end shoppers, be it a website, physical store or a mobile app. Using the correct website channel ensures that the correct branding and identity features are displayed to shoppers, enhancing their experience.

Release notes

📅 *Multiple channels and other channels beyond websites will be supported in future releases.*

☑Test case 5: Update and list a channel collections for an account

Channel collection information is crucial for enhancing the shopper's purchase and post-purchase experience. In a production environment, the customer will be able to take additional actions such as reaching out to customer support or reviewing return processes for the specific merchant where they have made a purchase. This information is linked to specific channels like websites, apps, or physical stores, ensuring that all transactions made through these channels have access to relevant support information when needed.

Release notes

📅 *Multiple channels and other channels beyond websites will be supported in future releases.*

☑Test case 6: Fetch and update account information

It's important that the account information reflects the latest information about your merchants at all times. Fetch and update account information to ensure that all systems are aligned with regards to this information.

This should be programmatically handled whenever an update is applied to an account in your system. Ensure you are testing that the end-to-end functionality is working as expected.

☑Test case 7: Fetch and update the business information for an account

Fetch and update the business information for an account to ensure that the details were correctly sent on onboarding, and that the details can be updated afterwards when required.

This should be programmatically handled whenever an update is applied to an account in your system. Ensure you are testing that the end-to-end functionality is working as expected.

Steps to follow:

1. Fetch the business information for an already existing account.
2. Update any of the details previously fetched for the same account.
3. Verify that the changes are correctly updated in Klarna's system.

☑Test case 8: Create, get, and delete a signing key for an account

Signing keys are used to verify webhook notifications.

Steps to follow:

1. Create a new signing key.
2. Delete the same signing key.
3. Verify webhooks are working or not, depending on the status of the signing key in use.

☑Test case 9: Error handling

Is input sent to Klarna validated, and how are potential errors displayed to the merchant? See the Error handling page for more info of error details and how to handle them.

End-to-end test cases

Test your Klarna integration by following the steps for each of the cases below. Here are some things to keep in mind when you test:

- You can verify the results in the Orders app in the Klarna test portal.
- In all test cases, use Klarna's sample shopper data for the market you are testing.
- Make sure to use your test environment API credentials. Your production keys won't work.
- You can also validate optional data using the Logs app in the Klarna test portal. Keep the API reference open as it will help you to understand the details in the logs.

Happy testing!

Security Warning

⚠️ *Don't use any real-life data when testing. Instead, use the sample* shopper *data and sample payment data provided here*.

Ecommerce end-to-end test cases:

☑Test case 1: Complete, fully capture, and fully refund a payment transaction

Steps to follow:

1. Complete a payment transaction with one item: validate your request and the responses received in your backend and check that the product name, price, tax amount, quantity and shopper details match the information provided during the customer flow.
2. Process a full capture: simulate shipping the goods to the shopper, verify the transaction has been captured in your system and in the Klarna test portal.
3. Process a full refund: simulate returning the transaction by refunding the amount back. Verify the transaction has been refunded in your system and in the Klarna test portal.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction is created in your system and in the Klarna test portal.
- The payment transaction details in the Klarna test portal and in your system match those entered during the test:
- The payment transaction status updates correctly for both capture and refund stages.

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases.*

☑Test case 2: Complete, fully capture, and partially refund a payment transaction

Steps to follow:

1. Complete a payment transaction with at least two items: verify that the cart is being updated when navigating between the checkout and the product pages.
   **Tip: Navigate back and forth between checkout and product pages, and add more than one item to the shopping cart everytime.*
2. Process a full capture: simulate shipping the goods to the shopper, verify the transaction has been captured in your system and in the Klarna test portal.
3. Process a partial refund: simulate returning just one item and verify that the payment transaction has a status Partially refunded in Klarna test portal, and that the refunded amount and the remaining open amount are correct.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly for both capture and refund stages.

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 3: Complete and partially capture a payment transaction, release the remaining authorization

Steps to follow:

1. Complete a payment transaction with at least two items.
2. Process a partial capture: simulate shipping just some of the purchased goods to the shopper, verify the transaction has been captured in your system and in the Klarna test portal.
3. Capture the payment in full: simulate shipping the goods to the shopper, verify the transaction has been captured in your system and in the Klarna test portal.
4. Process a partial refund: simulate returning just one item and verify that the payment transaction has a status Partially refunded in Klarna test portal, and that the refunded amount and the remaining open amount are correct.
5. Release the remaining amount: verify that the captured amount is correct in your system and in the Klarna test portal, and there's no remaining open amount for the transaction.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- Transaction details in the Klarna test portal match those entered during the test.
- Transaction status updates correctly for both capture and refund stages.
- Transaction has no remaining open amount left

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 4: Complete, fully capture, and fully refund a payment transaction with a discount code

Steps to follow:

1. Complete a payment transaction with at least two items (same as Test case 2) with a discount code.
2. Process a full capture (same as Test case 1): ensure the discount code is taken into account in your system and in the Klarna test portal.
3. Process a full refund (same as Test case 1): ensure the discount code is taken into account in your system and in the Klarna test portal.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly for both capture and refund stages.
- The discount is taken into account in the captured and the refunded amount.

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 5: Complete, fully capture, and partially refund a payment transaction with a gift card

Steps to follow:

1. Complete a payment transaction with at least two items (same as Test case 2) and a gift card to reduce the payment amount.
2. Process a full capture (same as Test case 1): ensure the discount of the gift card t is taken into account in your system and in the Klarna test portal.
3. Process a partial refund (same as Test case 3): ensure the discount of the gift card is taken into account in your system and in the Klarna test portal and the refunded amount and the remaining open amount to be paid are correct.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly for both capture and refund stages.
- The gift card is included in the line items

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 6: Complete and cancel a payment transaction

Steps to follow:

1. Complete a payment transaction with one item (same as Test case 1)
2. Cancel the payment: verify the cancellation in your system and in the Klarna test portal, check the transaction's status is Canceled.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly after canceling the transaction.

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 7: Complete a payment transaction with different customer and shipping details

Steps to follow:

1. Complete a payment transaction with one item (same as Test case 1): use different customer and shipping recipient details.
2. Verify that the transaction has the correct shipping details in your system, and in the Klarna test portal.

Expected outcome:

- The payment confirmation page loads correctly upon transaction completion.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly for both capture and refund stages.

Release notes

📅 *Payment transaction management API will be available via the Klarna Network APIs in future releases*

☑Test case 8: Complete a denied payment transaction

Steps to follow:

1. Complete a payment transaction with one item (same as Test case 1): use **denied test shopper details** that result in denied purchase.

2. Confirm the payment flow redirects back to the checkout page, and it allows you to choose another payment method.

Expected outcome:
- The shopper can change the payment method after the initial failed payment attempt.

☑Test case 9: Complete a free purchase transaction

Steps to follow:
1. Initiate a payment transaction where the total order amount is 0.
2. Proceed through the checkout process as a regular shopper.
3. Confirm that the payment flow completes successfully without requiring any payment method.
4. Verify that the order confirmation page is displayed, indicating a successful transaction.

Expected outcome:
- The shopper is able to complete the purchase without any payment method since the total order amount is 0.
- The order confirmation page is displayed, confirming the successful completion of the transaction.

☑Test case 10: Verify the purchase flow in desktop, mobile and app views

Steps to follow:
1. Complete a payment transaction with one item in desktop view (same as Test case 1)
2. Complete a payment transaction with one item in mobile view (same as Test case 1)
3. Complete a payment transaction with one item in mobile app (same as Test case 1)
    a. Confirm that any links redirecting to third party apps (e.g. Bank ID and other authentication apps) work.

Expected outcome:
- The payment confirmation page loads correctly upon transaction completion for all devices and views.
- The payment transaction details in the Klarna test portal match those entered during the test.
- The payment transaction status updates correctly for both capture and refund stages.

Sample shopper data and test triggers

This section contains sample data you can use to perform the testing of your integration in the Klarna test environment and complete the Test cases section in the Klarna playground environment.

Sample business data

In order to test the Management API, you need to type of business data:

- Business partner data: please reach out to your Klarna point of contact so they can share all parameters needed.
- Merchant account data: please use the following information to test merchant onboarding.

Account owner

Parameter	Sample
`given_name`	John
`family_name`	Doe
`email`	john.doe@example.com
phone	+18445527621

Business information

Parameter	Sample
`business_name`	John Doe LLC
`business_entity_tyoe`	LIMITED_LIABILITY_COMPANY
`registration_authority`	Ohio
`registration_name`	12345678
`tax_registration_ number`	999-999-999
`financial_registration_number`	123-456-789

Operating/registration address

Parameter	Sample
`street_address`	800 N. High St

`street_address2`	Ste. 400
`postal_code`	43215
`city`	Columbus
`region`	OH
`country`	US

⚠️If you decide to use any other data, do NOT use Personally Identifiable Information (PII).

Sample shopper data

To test the standard approved and denied payment flows in Klarna payments use this data.

Sample payment data

To test different payment methods use this data.

Considerations:

Every Klarna payment method features distinct thresholds based on market specifics that should be considered when testing.

These limits are subject to change without notice and should not be hardcoded.

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
United States of America	Min: 0 USD Max: 1000 USD	Min: 35 USD Max: 4000 USD	6 months 149-10000 USD 12 months 299-1000 USD 18 months 599-10000 USD 24 months 999-10000 USD	Min: 0 USD Max: 4000 USD	NA	NA
Australia	Min: 0 AUD Max: 500 AUD	Min: 35 AUD Max: 2000 AUD	NA	Min: 0 AUD Max: 4000 AUD	NA	NA
Austria	Min: 0.1 EUR Max: 5000 EUR	Min: 25 EUR Max: 5000 EUR	6 months 25-10000 EUR 12 months 120-10000 EUR	Min: 0 EUR Max: 10000 EUR	Min: 0 EUR Max: 5000 EUR	Min: 0.1 EUR Max: 14000 EUR

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
			18 months 1000-10000 EUR 24 months 1000-10000 EUR			
Belgium	Min: 1 EUR Max: 1500 EUR	NA	NA	Min: 0 EUR Max: 10000 EUR	Min: Max:	Min: 0.1 EUR Max: 14000 EUR
Canada	NA	Min: 35 CAD Max: 1500 CAD	NA	Min: O CAD Max: 2000 CAD	NA	NA
Denmark	Min: 1 DKK Max: 50000 DKK	Min: 350 DKK Max: 50000 DKK	NA	Min: 0 DKK Max: 100000 DKK	NA	NA
Finland	Min: 1 EUR Max: 5000 EUR	Min: 25 EUR Max: 5000 EUR	6 months 25-5000 EUR 12 months 120-5000 EUR 18 months 240-5000 EUR 24 months 360-5000 EUR	Min: 0 EUR Max: 10000 EUR	NA	Min: 0.1 EUR Max: 14000 EUR
France	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1500 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Germany	Min: 0.1 EUR Max: 10000 EUR	Min: 25 EUR Max: 10000 EUR	6 months 25-10000 EUR 12 months 120-10000 EUR 18 months 1000-10000 EUR 24 months 1000-10000 EUR	Min: 0 EUR Max: 10000 EUR	Min: 0 EUR Max: 5000 EUR	Min: 0.1 EUR Max: 14000 EUR
Italy	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1500 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Netherlands	Min: 1 EUR Max: 5000 EUR	Min: 35 EUR Max: 4000 EUR	NA	Min: 0 EUR Max: 10000 EUR	Min: 0 EUR Max: 5000 EUR	Min: 0.1 EUR Max: 14000 EUR
New Zealand	NA	Min: 35 NZD Max: 2000	NA	NA	NA	NA

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
		NZD				
Norway	Min: 1 NOK Max: 150000 NOK	Min: 250 NOK Max: 150000 NOK	6 months 250-150000 NOK 12 months 1200-150000 NOK 18 months 2400-150000 NOK 24 months 3600-150000 NOK	Min: 0 NOK Max: 100000 NOK	NA	NA
Poland	Min: 0 PLN Max: 7000 PLN	Min: 150 PLN Max: 5000 PLN	NA	Min: 0 PLN Max: 20000 PLN	NA	NA
Spain	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1500 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Sweden	Min: 1 SEK Max: 150000 SEK	Min: 300 SEK Max: 849.99 SEK *These thresholds may vary depending on the agreement	6 months 250-150000 SEK 12 months 1200-150000 SEK 18 months 2400-150000 SEK 24 months 3600-150000 SEK	Min: 0 SEK Max: 100000 SEK	Min: 0 SEK Max: 50000 SEK	Min: 1 SEK Max: 150000 SEK
Switzerland	Min: 1 CHF Max: 2500 CHF	NA	NA	Min: 0 CHF Max: 10000 CHF	NA	Min: 0.1 CHF Max: 13000 CHF
United Kingdom	Min: 1 GBP Max: 600 GBP	Min:30 GBP Max: 2000 GBP	6 months 250-5000 GBP 12 months 500-5000 GBP 18 months 1200-5000 GBP 24 months 1200-5000 GBP	Min: 0 GBP Max: 4000 GBP	NA	Min:0.1 GBP Max: 115000 GBP
Ireland	NA	Min: 35 EUR Max: 1500 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
Portugal	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1000 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Mexico	NA	Min: 700 MXN Max: 20000 MXN	NA	NA	NA	NA
Romania	NA	Min: 200 RON Max: 5000 RON	NA	Min: 0 RON Max: 20000 RON	NA	NA
Greece	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1000 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Czech Republic	NA	Min: 850 CZK Max: 25000 CZK	NA	Min: 0 CZK Max: 100000 CZK	NA	NA
Hungary	NA	Min: 14000 HUF Max: 400000 HUF	NA	Min: 0 HUF Max: 1500000 HUF	NA	NA

Resources

Klarna integration principles

When designing solutions and integrating with Klarna APIs, think about long-term performance and scalability from the very beginning. It is critical to craft a scalable architecture capable of supporting growth and ensuring efficient API use as well as enabling regular monitoring and agile responsiveness are essential for maintaining peak performance.

Our solutions are designed around the following principles in mind: :

- **Dynamic product availability**: Confirm product availability dynamically and retrieve payment descriptors. This approach allows to tailor the shopper experience to increase conversion rates as well as enables flexibility to accommodate future product developments, reducing development time when entering new markets or launching new products. It also enhances compliance with regulatory changes.
- **Idempotency and fallback logic**: Implement idempotency wherever possible to ensure consistency between systems. Establish fallback logic to synchronize updates and system alignment in case of incidents or delays. This strategy improves recovery time in the event of a disruption.
- **Embrace continuous improvement:** Regularly test, learn, and refine your integration to keep pace with shopper expectations and industry developments. Stay updated with the latest features from Klarna to further improve the user experience.

Further recommendations and best practices are shared throughout this documentation. By adopting a user-focused, security-conscious, and performance-driven approach and committing to continuous improvement, you can develop a Klarna solution that not only fulfills your business goals but also delights your shoppers.

Klarna ecosystem

Environments

Klarna provides both test and live environments, each designed to support seamless global integration of the Partner Management API, Partner Product API, and other Klarna products, irrespective of your location, the shopper's origin or other particular considerations.

Klarna Partners are required to make both test and live environments available to all Partners integrated via their services to allow for the validation of their Klarna integration. All services available in production are required to be included in this availability. Klarna requires that accounts in any environment not be shared across multiple Partners.

These environments function entirely independently of each other, and may behave differently as a result of the below considerations:

- **Access and authentication**: Different base URLs and credentials are used to access the live and test environments.
- **Functionality**: The test environment simulates the live environment but lacks active fraud assessments, including any identity or address detail validation. Any atypical flow needs manual initiation through test triggers.
- **Data security**: The test environment does not use One-Time Passwords (OTP), making it inappropriate for sharing Personal Identifiable Information (PII). PII entered into the test environment is replaced with synthetic data, which means response values might vary unless Klarna's sample data is used. This approach minimizes data leak risks.
- **Settings**: Adjustments made in one environment do not affect the other. For instance, changes to branding or product offering made in the test environment won't impact the live environment.
- **Transactions and accounts**: Transactions or user accounts created in one environment do not transfer to the other. For example, transactions placed in the test environment will not appear in the live environment.

Endpoints

Global base URLs are provided to optimize latency and enhance fault tolerance.

Environment	DNS	IP Addresses
Live	`api-global.klarna.com`	`13.248.252.240, 76.223.28.105`
Test	`api-global.test.klarna.com`	`3.33.145.71, 13.248.213.183`

Callbacks

Callbacks from Klarna will originate from specific IP addresses for each environment.

Environment	Callback IP Addresses
Live	`52.17.117.56, 52.17.176.198, 52.0.45.33, 52.0.46.187, 13.211.30.100, 3.104.49.49, 13.54.229.130`
Test	`34.242.203.160, 34.242.19.4, 52.45.47.152, 34.235.91.238, 3.24.91.202, 52.62.115.68, 52.63.129.92`

Versioning and deprecation

Klarna is committed to ensure that updates to our APIs are backward compatible whenever possible, allowing your systems to continue running smoothly as new features are introduced. Should there be any breaking changes, these will be implemented under a new API version.

Examples of breaking changes:

- **Removal of ENUM values**
- **Removal of actions** (HTTP Methods)
- **Removal of resources**/Endpoints
- **Change in state machine transitions**, or introduction of new states

Backward-compatible changes

We classify certain updates as backward-compatible, meaning they should not require modifications on your side to continue using the API effectively.

Your integration should be capable of handling the following changes:

- **Adding new API resources**: Introduction of new endpoints or resources will not affect existing functionality.
- **Adding new optional request parameters**: New parameters added to existing API methods will not alter the behavior of existing calls; they will provide additional functionality if you choose to use them.
- **Adding new properties to API responses**: Additional fields in API responses are designed to be ignored by partners who do not expect them, ensuring compatibility with older versions.
- **Changing the order of properties**: The sequence of properties in API responses may vary, but this will not impact integrations that rely on proper key-based parsing instead of the order of data.
- **Adjustments to opaque strings**: Changes to the format of strings such as IDs.
- **Introducing new event types**: New events might be added to our webhooks, if they are optional, and do not affect the behavior of the existing integration. Ensure your webhook listeners are prepared to handle atypical event types gracefully, either by logging them for review (recommended) or safely ignoring them.

Best practices for ensuring a high-quality integration

To optimize your integration and prepare for future updates, consider these best practices:

- **Flexible data handling**: Implement flexible data parsing that can accommodate additional fields without causing errors or disruptions.

- **Regular updates**: Stay updated with our latest API documentation and changes. Regular updates to your integration can help leverage new features and enhancements while maintaining compatibility. Klarna will keep you informed about deprecations.

- **Error handling**: Develop robust error handling mechanisms to manage unexpected API responses or failures gracefully. This minimizes the impact on the user experience and makes your application more resilient.

By adhering to these guidelines and preparing your integration for both backward-compatible changes and keeping your integration up-to-date with the latest API version from Klarna will ensure a seamless interaction with Klarna's evolving API landscape.

Availability and latency

In this section you will find details of service level commitments related to Klarna's solutions. This includes the execution of API calls related to the creation of new transactions from an accounts website through Klarna's API as well as other features and functionalities that may be provided as part of our services.

Latency

The latency of a service indicates the time to get a response to a request done to Klarna service. This latency is measured and calculated on all requests at the 99 percentile and at the edge of the region in which the service is deployed. This latency however, does not include Internet network latency impact.

Downtime

Downtimes in Klarna's services reflect the actual time when a Klarna solution is not responding to requests with status code being 2xx, 3xx or 4xx. It is important to note that the following scenarios are not considered downtime:

- Unavailability due to circumstances that are outside of Klarna's control such as force majeure which affects all of Klarna's redundant and geographically dispersed production sites.

- Any unavailability or downtime attributable to acts or omissions of Acquiring partners, Third Party Payment Option Providers, banks or other external data providers.

- Unavailability caused by or attributable to the Partner and/or any of the Partners contractors, suppliers or any other third party that the Merchant cooperates with.

- Unavailability due to maintenance downtime. Transactional services are designed to be zero downtime, however in the exceptional case that maintenance downtime is required, Klarna will inform this via our Status monitor system with at least 7 days in advance.

Technical Support

Customer support for merchants is available by email, chat or telephone from 9:00 to 17:00 local time on business days (Monday to Friday).

Weekend availability is based on the market and may vary, check specific market availability here.

Web SDK

Klarna.js is a Web SDK that bundles all our products, including payments and boost products.

This SDK will allow you to handle your most frequent use cases easily, with a few lines of code, and allowing customization for additional edge cases.

Our API is primarily redirect driven, but will run on-page through modal when supported by the device. The same integration pattern can be used for both redirect and on-page mode. This means that the SDK survives a loss of JavaScript context and can restart itself.

To learn more on how to integrate Klarna leveraging Klarna.js, see Recommended integration: Klarna.js section.

Consult the SDK reference for a complete description of the specifications.

Mobile SDK

Klarna Mobile SDK enables Klarna services to work seamlessly within a native mobile app.
The Mobile SDK supports Klarna Payment, Klarna Express checkout, Sign in with Klarna, On-Site messaging, and more, using technologies like Kotlin, Swift, JavaScript, and Typescript.

This SDK is the recommended default way to use Klarna products in mobile applications. This is mainly due to the limitations of the WebViews in both iOS and Android. The SDK adds iOS/Android native components and lets Klarna services overcome those issues with a communication between web and native environments.

Not using the Mobile SDK can degrade your Klarna integration and may prevent customers from completing their payments because:

- Third-party banks and card processors may block or restrict interactions through WebViews.
- Cookies in WebViews are handled differently, causing additional friction during checkout.
- WebViews don't handle navigation to third-party applications for authorization and authentication, while the Mobile SDK does.
- The App Handover feature enabled by Mobile SDK allows seamless redirection for authentication and consent within the Klarna app, enhancing security and streamlining processes.
- The Mobile SDK supports SSO/"Remember me" across apps, enabling shared login and pre-filled shopper details, making it easier for returning users.

Security

API Authentication Standards

All server-side REST APIs require API keys for authentication, whereas the Klarna Web SDK uses Client IDs. Ensure all API requests are transmitted over HTTPS using TLS 1.2 protocol at a minimum. Attempts to connect without valid credentials or via plain HTTP will not succeed.

API keys are sensitive; handle them with utmost care.

The TLS certificates at API endpoints are issued by AWS Certificate Manager and are subject to automatic renewal as expiration approaches. We advise against reliance on specific certificate details, recommending instead trust in the root CA as outlined in the documentation.

Global authentication for Partner Product API

For acquiring partners working with multiple account_ids, you should consider the following:

- `account_id` must be included in the path for operations on a specific merchant or its resources (use the `account_id` returned by the Management API):
    - `/v1/accounts/`**`{account_id}`**`/payment/requests`
    - HTTP header :
    - Authorization: `Basic <api_key>`

Release notes

📅 This will be available in future releases, in the meantime, please ensure the following:

- `Klarna-Partner-Account` HTTP header must be included for operations on a merchant or its resources (use the `account_id` returned by the Management API):
    - `/v1/accounts/`**`{account_id}/`**`payment/confirmation-tokens/{payment_confirmation_token}/confirm`
    - HTTP header:
    - Authorization: `Basic <api_key>`

DDOS Protection

Our integration APIs are fortified with active DDOS protection measures designed to stop traffic identified as illegitimate or exhibiting atypical behaviors. If a DDOS protection rule is triggered, the HTTP-status code 403 will be returned, absent the typical error information object.

Further information about rate limiting is available in the Rate limiting section.

Communication security

The global API endpoint is secured via 2 anycast static IPs, enabling partners to configure egress security measures within their IT infrastructure effectively.

API key usage can be restricted to designated CIDR blocks, ensuring only authorized calls from predetermined IP addresses are allowed. This measure effectively restricts access to Klarna's API to trusted networks, reducing the risk of unauthorized access.

Callbacks from Klarna will originate from specific IP addresses based on the environment, information which should be used to configure firewalls for enhanced security.

Security Protocols and Best Practices

Security protocols vary by integration and should be assessed individually. However, some universal requirements include:

- Maintaining up-to-date security across all system components, promptly applying the latest patches, and employing a thorough testing process before deployment.
- Carrying out regular fraud assessments to pinpoint and address potential security issues.
- Limiting administrative rights strictly to those who need them, adhering to the principle of least privilege.
- Keeping a formal log of all individuals with access to Klarna systems and ensuring access is granted via corporate email addresses.
- Regularly monitoring and updating access rights, especially after an employee's role changes or departure, and conducting periodic access reviews.
- Avoiding shared accounts to ensure actions can be attributed to individual users.
- Enforcing the use of strong passwords (14 or more characters) and enabling two-factor authentication (2FA) where feasible.
- Encrypting stored secrets and not keeping them in plaintext.
- Considering suppliers and third-party providers within the organization’s overall security strategy and conducting appropriate evaluations.
- Enabling logging for sensitive actions and monitoring for suspicious activities.

Klarna mandates that partners report any suspicious activities through partner support or the Klarna portal chat. This includes unusual Klarna transaction processes. Such collaborative vigilance is crucial in detecting and mitigating potential threats early, ensuring a secure environment for all parties involved. Klarna reserves the right to disable API keys upon detecting any evidence of potential compromise.

Adherence to these guidelines is essential for maintaining robust security standards and protecting against potential vulnerabilities.

Authentication type by service

Two authentication methods are used in the platform: API authentication via an **API key** and a **client ID**, employed to authenticate the calling account.

- The **API key** is highly confidential and must never be exposed in clear-text beyond an API request.
- The **client-id** is used in a browser environment and is not secret in itself, it must be configured with a list of approved websites from which it's approved to be used which will prevent some fraud scenarios

Both API-keys and client-ids are signed tokens which are verified by the platform to ensure the integrity of the information.

The pattern for both API-keys and client-ids are: `klarna_<api/client>_<live/test>_<token>`

Client ID Structure:
```
klarna_<live|test>_<client>_<random>
```
Client ID Example:
```
klarna_test_client_elZGI1B5dHBIRWcjZrNldnbEVj[...]uefnc3
```
API Key Structure::
```
klarna_<live|test>_<api>_<random>
```
API Key Example:
```
klarna_live_api_elZGI1B5dHBIRWltRjF5cjZrNldnbEVjKnIqeC[...]Uybz0
```

The authentication information used by features of the platform

Feature	Authentication type
Web SDK	Client-id
REST API	API-key
Sign-in-with-klarna	OAuth using client-id and API-key

Rate limiting

To safeguard our APIs from potential misuse due to coding errors, suboptimal integrations, and malicious activities, we implement proactive rate limiting. This document outlines the classifications, enforcement, and management of rate limits across various API categories critical to our services.

API operation categories

We classify API actions based on their relevance to the purchase process and the resources they consume. This classification helps minimize interference between processes, ensuring efficient operation. For example, we strive to prevent extensive settlement report processes from impacting new payment transaction capabilities.

API rate limit action categories are as follows:

- **Account onboarding**: Involves resource-intensive tasks such as creating accounts, requiring synchronous calls to other APIs.
- **Payment transaction capture**: Crucial actions for capturing payment transactions.
- **Payment transaction management**: Covers actions related to managing payment transactions that are not essential for the capture process.
- **Settlement**: Includes actions that could affect rate limits in other areas, identified through access log analysis.
- **Dispute**: Specifically addresses operations related to handling disputes.
- **Partner management**: Encompasses workflows for managing partners, excluding the onboarding of new partner accounts.

Rate limit enforcement

Requests to the global API endpoint are processed in the closest data center relative to the caller's location. The rate limit configuration for a given Partner and Acquiring Partner is the same across all locations within a specific environment. However, the rate limit quota is enforced by each data center. Therefore, requests for the same sub-partner from different parts of the world may experience different rate limit statuses.

The rate limit mechanism tracks rate limits for API operations at two distinct levels:

- Acquiring Partner level: This encompasses all operations, including those on sub-partners.
- Partner level: These limits apply uniquely to each sub-partner.

If either limit is reached, a request will be subject to rate limiting.

API Category	Partner	Acquiring Partner
Rate Limit by API Operation Category in Production		
account-onboarding	0/s	10/s
payment-transaction-capture	50/s	200/s
payment-transaction-management	200/s	500/s
dispute	20/s	50/s
settlement	0/s	150/s
partner-management	20/s	50/s
Rate Limit by API Category in Playground		
account-onboarding	0	5/s
payment-transaction-capture	12/s	50/s
payment-transaction-management	50/s	125/s
dispute	5/s	12/s
settlement	0/s	37/s
partner-management	5/s	12/s

Handling rate limits

A rate-limited request returns an HTTP-status code 429 and headers indicating the remaining quota:

- `X-Ratelimit-Limit`: Information on the rate limits and the metering interval. The first item is the quota that is closest to being exceeded. This is followed by one or more rate limit quota policy descriptions.
- `X-Ratelimit-Reset`: Time in seconds until the current metering interval resets.. For per-second rate limiting, this will always be “1”.
- `X-Ratelimit-Remaining`: The approximate remaining quota of the rate limit

When a rate limit is reached, we require implementing a retry mechanism with exponential back-off to prevent retry attempts from retriggering rate limiting; add jitter to the back-off as retrying all attempts together is likely to draw out the issue. In addition, Klarna suggests that a token bucket algorithm is used to control the global flow of requests from the calling system to the API, this will help to reduce the likelihood of rate limiting in future.

Rate limit quota policy

A rate limit quota policy element is describing a rate limit that has been evaluated for the request. More than one can be active at the same time. In the current API there will be two, one for the Acquiring Partner-level rate limit and one for the sub-partner ratelimit.

Example of a single quota policy:

```javascript
40;w=1;name="ratelimit-name"
```

- The “40” is how many units are available within a refresh-interval
- The “w=1” describes the length in seconds of interval after which the rate limit quota is reset
- The “name” component is optional and should be seen as informational and can change without notice. Typical values for the name will include the level where the rate limit applies and the rate limit operation category.

Example:
The values in the headers are approximate and provided on a best-effort basis.

Example:

```javascript
X-Ratelimit-Limit: 40,
40;w=1;name="account_payment-request-capture",100;w=1;name="psp_payment-request-
capture"
X-Ratelimit-Remaining: 15
X-Ratelimit-Reset: 1
```

The information returned with the above response should be interpreted as follows:

- The rate limit quota closest to being reached for the interval is 40 requests per second with the descriptive name “`account_payment-request-capture`”
- The quota window interval is 1 second
- There are 15 requests remaining within the 1-second time window before further requests are rejected
- There is 1 second until this quota resets

Common causes for rate limiting

We aim to set rate limiting quotas to ensure that the majority of merchants experience no rate limitations for legitimate traffic. However, you may encounter rate limits in the following scenarios:

- **High request rate**: Exceeding the defined rate limits due to a rapid influx of requests within a short timeframe may trigger rate limiting. To address this, it is advised to distribute requests evenly over the interval specified by informational headers and employ retries with exponential backoff.

- **Traffic surges**: A sudden and substantial increase in traffic, such as during a flash sale, can deplete the rate limit quota. If you anticipate an upcoming event that may surpass your request quota, kindly reach out to Partner Support or contact your Account Manager for assistance.

- **Bot-generated requests**: The presence of bots on a website may result in an increased frequency of requests, potentially leading to rate limiting as a preventive measure against web scraping or data harvesting. It is recommended to implement bot-detection methods and, if suspicious bot activity is detected, initiate authentication before making API calls.

Rate limiting change management

Adjustments to rate limiting protocols are managed per the Klarna change management process as outlined in Versioning and deprecation. Changes may occur without prior notice in response to abuse or consistent deviation from best practices.

It is crucial to monitor the rate limiting information returned in response headers to adapt to any adjustments effectively.

Integration resilience

Idempotency

Idempotency is a key concept in system operations, ensuring that repeating the same action multiple times doesn't change the outcome after the first execution. This principle is crucial for maintaining consistency and reliability, particularly in payments integrations, enhancing user experience and system stability.

Klarna requires idempotent integration of its systems for actions that could change a transaction's status. This safeguards against unwanted changes or duplications if a request is repeated. To manage this, Partners can use the ‘`Klarna-Idempotency-Key`’ header in all POST and PATCH requests. An idempotency key should be created using the UUIDv5 standard, and is valid for 24 hours - outside of that window Klarna cannot guarantee the idempotency key will be honored with respect to an action.

This enables Klarna to recognize and ignore repeat requests to ensure an action is not unintentionally duplicated. In the case of a duplicate attempt, Klarna will respond with the initial result instead of processing a new one.

Tagging

Integration tagging is a crucial mechanism for ensuring seamless interoperability across various platforms and Acquiring Partners. This system improves the integration experience by providing detailed and consistent data to support customer experience, performance tracking, and incident resolution.

All operations and interactions with Klarna should be associated integration partners to ensure maximum monitorability and performance. To provide this level of detail, Klarna requires that partners provide all available information with each interaction.

Data requirements

- **Integration metadata:** Present in all API/SDK requests sent to Klarna, this includes:
    - **Name:** Identifies the specific integration pathway used by the merchant.
    - **Version:** Tracks the version of the integration pathway to ensure data accuracy and facilitate troubleshooting.
    - **Upstream integration metadata:** Represents an array of integrations used by a given merchant within the hosting partner (e.g., a PSP), including name and version details. This contains the name and version of each of these sub-integrations.
- **Platform transaction context:** Included in API/SDK requests during payment interactions, this encompasses:
    - **Identifier:** The specific identifier of the platform used by the merchant to make the request, ensuring traceability.

- **Platform reference:** A unique transaction identifier assigned by the platform, linking a shopper across the entire journey.

Release notes

📅 Integration tagging will become available in subsequent releases.

Monitoring and alerting

To ensure compliance with integration best practices and data protection regulations, the Partner must proactively monitor and share information about deviations from expected behaviors as outlined in this and/or the Partner-specific Solution Scope Document. This includes technical errors and unusual activities by shoppers or, where relevant, accounts or integrations occurring through your integration.

To support monitoring, Partners are required to meet the following criteria:

- Immediately escalate any events that disrupt business operations, compromise the integrity or security of information systems, or impact the availability, confidentiality, or integrity of digital assets.
- Address any disruptions or compromises affecting the operation and reputation of the Klarna payment system.
- Klarna Partners must ensure that all integration approaches must include a specific parameter that uniquely identifies the specific integration being utilized on that request or transaction. This parameter must be traceable across all integrations to ensure comprehensive incident handling, behavioral tracking, and account management.
- Follow a release process that validates system functionalities and integration points in the test environment to detect and resolve issues before they impact system performance or shopper experience.
- All interactions with Klarna are tagged with the appropriate integration details and versions as defined in Integration tagging

Error handling

When an error occurs on API request, Klarna responds with an error type, an error code, an error message and a corresponding HTTP status code.

Klarna's APIs use HTTP status codes together with error objects to handle errors. When an API call fails Klarna will respond with a 4xx or 5xx status code together with a response body containing an error object with the error code, an array of error messages and a unique error id to be used to identify the request.

Descriptions for the response fields:

Parameter	Definition
`error_id`	A unique identifier for the request generated by Klarna. This ID will help you in investigations in case you need help from our support team.

`error_type`	Type of the error. Different error types are ACCESS_ERROR, TECHNICAL_ERROR, RESOURCE_ERROR and INPUT_ERROR.
`error_code`	Error code for further categorizing the error. We recommend using this error code for building your error handling logic.
`error_message`	A human readable error message. The error message is not meant to be displayable to end-users, but to assist in technical troubleshooting.
`doc_url`	Link to Klarna docs describing how to use the API to avoid the error, or a more detailed explanation of why the error occurred. To be provided when available.

Example of API response with an error details:

```
Unset
{
  "error_id": "abcd1234-12ab-1234-abcd-abcd12345678",
  "error_type": "INPUT_ERROR",
  "error_code": "VALIDATION_ERROR",
  "errors": [
    {
      "parameter": "line_items[0].quantity",
      "error_message": "Parameter line_items[0].quantity must be greater than or equal to
1",
      "doc_url": "https://docs.klarna.com/....."
    }
  ]}
```

Defined error types, and how to handle them

Error Type	Error Code	HTTP Status Code	Definition	Handling
	NOT_FOUND	404	The requested API route does not exist.	Verify the API route.
ACCESS_ERROR	UNAUTHORIZED	401	The presented credentials failed authentication.	Verify the credentials, check the authentication method, and confirm the correct endpoint.
	RATE_LIMITED	429	The caller was rate limited due to too many requests.	See Rate Limiting page (LINK) for more details.
RESOURCE_ERROR	RESOURCE_NOT_FOUND	404	The resource was not found.	Verify the token provided in the request path. This

				issue may also occur if an attempt is made to update an expired shopping session.
	RESOURCE_CONFLICT	409	There was a conflict in using the resource.	The transaction details are conflicting with the request details. Might occur due to concurrent updates to the resource.
	OPERATION_FORBIDDEN	403	The provided credentials (authorization) does not have enough privileges to perform the requested operation.	Ensure that the credentials being used have the necessary permissions for the operation.
	RATE_LIMITED	429	The specific resource was rate limited. For example creation of resources are rate limited, but it is still possible to run on already existing resources.	See Rate Limiting page (LINK) for more details.
INPUT_ERROR	VALIDATION_ERROR	400	One or more input parameters failed input validation. For example exceeding a max value, or failed pattern matching, invalid type.	Verify that the request details and formats are correct. See the error message for more info.
	INVALID_CONTENT_TYPE	400	The input does not conform to the expected content type syntax. For example invalid JSON.	Verify that the content type is as expected.
TECHNICAL_ERROR	INTERNAL_ERROR	500	An unknown error occurred.	Reach out to your support contact and include the error message and the error id.
	TEMPORARY_UNAVAILABLE	503	The system is temporarily unavailable to process the request.	Reach out to your support contact and include the error message and the error id.