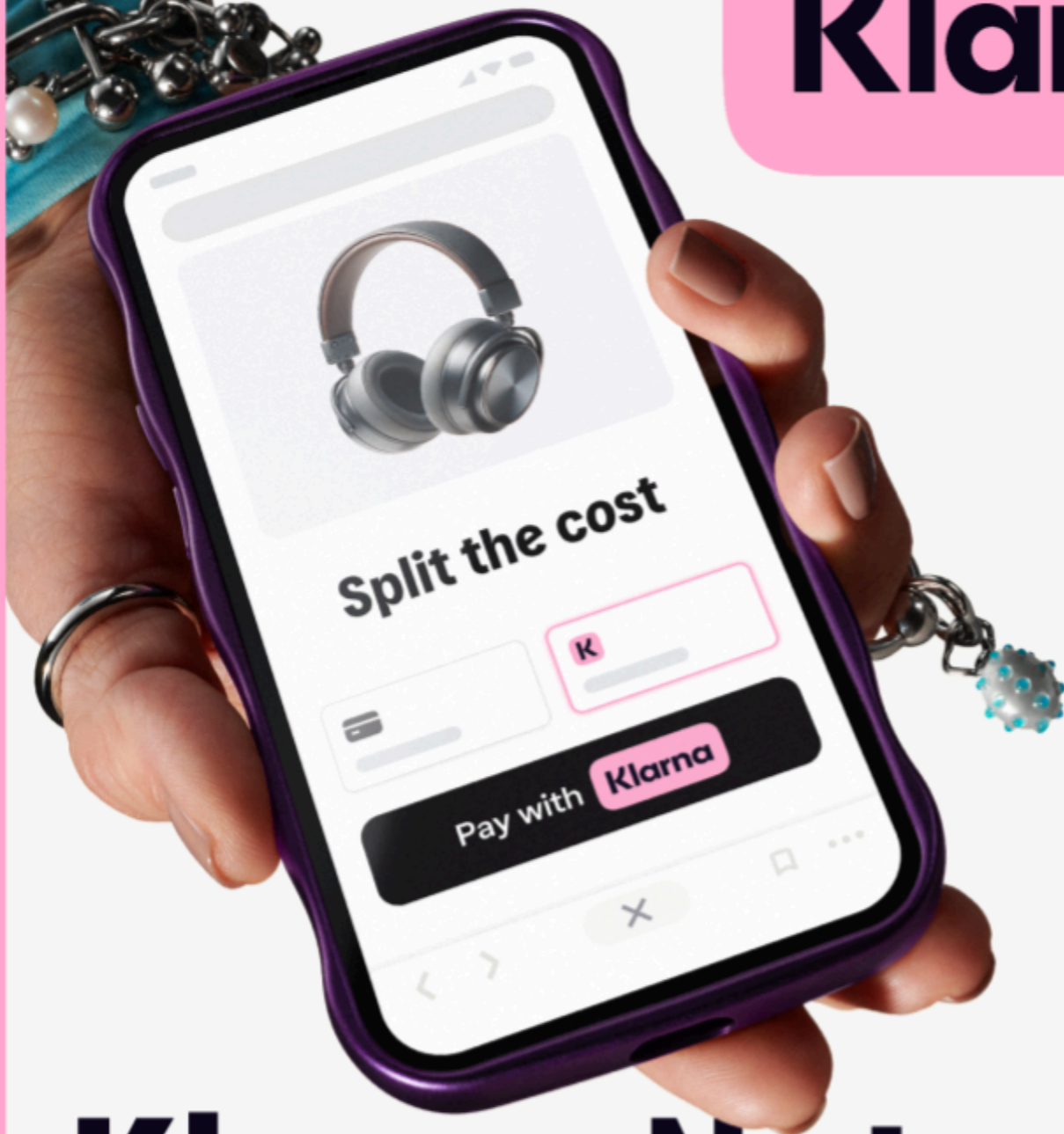


Klarna



Klarna Network

Integration Guidelines

Table of Contents

1. Introduction

- 1.1 Confidentiality disclaimer
- 1.2 How to use this document
- 1.3 Definitions
- 1.4 Understanding Klarna
 - 1.4.1 Smart solutions to maximize sales

2. Provide Klarna to my Partners

- 2.1 Before you start
 - 2.1.1 APIs overview
 - 2.1.2 Integration overview
- 2.2 Design your Klarna Solution
 - 2.2.1 Klarna product suite interoperability
- 2.3 How to present Klarna in the checkout
 - 2.3.1 Checking Klarna availability
 - 2.3.2 Checkout structure
 - 2.3.3 Payment flow
- 2.4 Set up your partner account
 - 2.4.1 Partner account
 - 2.4.1 Step 1: Configure account credentials
 - 2.4.2 Step 2: Configure Klarna webhooks
 - 2.4.3 Step 3: Account configuration management
- 2.5 Onboard and manage Partners
 - 2.5.1 Step 1: Determine account structure
 - 2.5.2 Step 2: Onboard Partners
 - 2.5.3 Step 3: Manage Partner payment products
- 2.6 Processing payments with Klarna Payment Services
 - 2.6.1 Online store transaction
 - 2.6.2 Tokenized Payments
 - 2.6.3 More payment use cases
 - 2.6.4 Important payment concepts
- 2.7 Interoperability
 - 2.7.1 Step 1: Grant access to Klarna's Partner portal
 - 2.7.2 Step 2: Consume Klarna Interoperability Token
 - 2.7.3 Step 3: Consume Klarna Interoperability Data
 - 2.7.4 Step 4: Consume Klarna Payment Confirmation Token and confirm Klarna payments
 - 2.7.5 Step 5: Consume Klarna Customer Token
- 2.8 Managing Payment Transactions
 - 2.8.1 Read and update payment transactions
 - 2.8.2 Capturing a Payment Transaction



- 2.8.3 Refund payment transactions and allocation
- 2.8.4 Void payment transaction
- 2.9 Disputes handling
 - 2.9.1 Common dispute use cases
 - 2.9.2 Managing Disputes
- 2.10 Pricing and reconciliation
 - 2.10.1 How pricing works
 - 2.10.2 Reconciling Klarna Settlements
 - 2.10.3 Settlement File
 - 2.10.4 Cut-off times for different regions
- 2.11 Test your integration

3. Test cases

- 3.1 Management API test cases
- 3.2 End-to-end test cases
 - 3.2.1 Online store end-to-end test cases
 - 3.2.2 Mobile app test cases
 - 3.2.3 Customer token end-to-end test cases:
- 3.3 Sample customer data and test triggers
 - 3.3.1 Sample business data
 - 3.3.2 Sample customer data
 - 3.3.3 Sample payment data
- 3.4 Create public documentation for your Partners
- 3.5 Integration checklist - Go-live
 - 3.5.1 Account - Management API
 - 3.5.2 Partner product API
 - 3.5.3 UX - Customer flow
 - 3.5.4 Reflect your partnership with Klarna
 - 3.5.5 Set up your production environment

4. Resources

- 4.1 Klarna integration principles
- 4.2 Klarna ecosystem
 - 4.2.1 Environments
 - 4.2.2 Versioning and deprecation
 - 4.2.3 Availability and latency
 - 4.2.4 Web SDK
 - 4.2.5 Mobile SDK
- 4.3 Security
 - 4.3.1 API Authentication Standards
 - 4.3.2 DDOS Protection
 - 4.3.3 Communication security
 - 4.3.4 Mutual Transport Layer Security
 - 4.3.5 Security Protocols and Best Practices
 - 4.3.6 Authentication type by service



- 4.4 Rate limiting
 - 4.4.1 API operation categories
 - 4.4.2 Rate limit enforcement
 - 4.4.3 Handling rate limits
 - 4.4.4 Rate limiting change management
- 4.5 Integration resilience
 - 4.5.1 Idempotency
 - 4.5.2 Tagging
 - 4.5.3 Monitoring and alerting
 - 4.5.4 Error handling

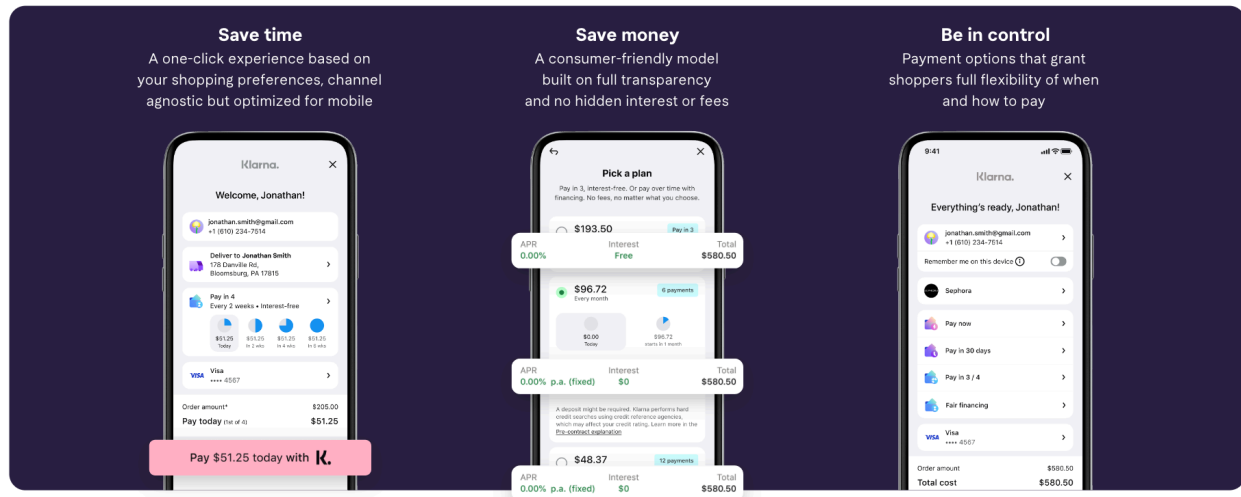
6. Annex

- 6.1 Changelog
- 6.2 Credential Creation
- 6.3 Manual Checkout (not for PSPs)
 - 6.3.1. Use the klarna.js public endpoint in the Payment package:
 - 6.3.2 Handle the button click event.



1. Introduction

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 helps our customers.



Klarna Payment Services offer a range of debit and credit programs, along with features that enable customers to make purchases online, in physical stores, and via mobile apps. The Klarna Network Rules are the set of terms and conditions that govern the use of Klarna Payment Services.

Within the Klarna Network, Partners have the flexibility to choose to integrate Klarna Payment Services through their preferred licensed and regulated entities authorized to distribute these Services, known as Acquiring Partners.

1.1 Confidentiality disclaimer

This document and the information in it are provided for the sole purpose of exploring potential business opportunities between you and Klarna. The document is the property of Klarna and is strictly confidential. The document contains confidential information that is intended solely for the person to whom it is transmitted. The disclosure of this document shall in no way imply any transfer or grant of rights to Klarna's confidential information, and Klarna retains all of its rights therein.

1.2 How to use this document

Who is the target audience?

The Klarna Integration Guidelines are intended for technical personnel at Klarna Partners and acquiring partners who are involved in implementing or enhancing a Klarna Network Integration. This document acts as a supplement to the Klarna Network Rules, [API specifications](#), and respective Klarna Network Distribution Agreements.

By adhering to the guidelines outlined in this document, all Klarna Partners contribute to a reliable and secure network, benefitting all users of Klarna's Product Suite. Adherence to these guidelines is crucial for upholding trust, security, and operational excellence across the network.



How should this content be interpreted and implemented?

The Klarna Integration Guidelines are designed to guide you through implementing Klarna in a way that meets all requirements. These guidelines are closely aligned with the Klarna Network Rules, by adhering to these guidelines, you will satisfy all integration requirements.

For partners unable to support the recommended integration path, alternative integration flows are provided. Each alternative path includes clearly defined criteria that must be met. All recommendations within this document should be treated as mandatory.

Is this document part of an agreement? Is it an attachment?

Klarna Integration Guidelines is a supplemental document meant to assist Klarna Partners in fulfilling the rules outlined within the Klarna Network Rules, and to ensure they achieve a best-in-class integration of Klarna products.

How can you use Klarna Integration Guidelines?

The Klarna Integration Guidelines may only be copied or adapted by an Acquiring Partner for the benefit of its Partners with [Klarna's explicit consent](#). This ensures that any dissemination of the guidelines preserves their original intent and integrity.

When are the Klarna Integration Guidelines updated?

Klarna periodically updates the Klarna Integration Guidelines to ensure accuracy, and operational efficiency. For detailed information on the schedule, process, and communication of these updates, please refer to the Klarna Network Rules.

When was this document last updated?

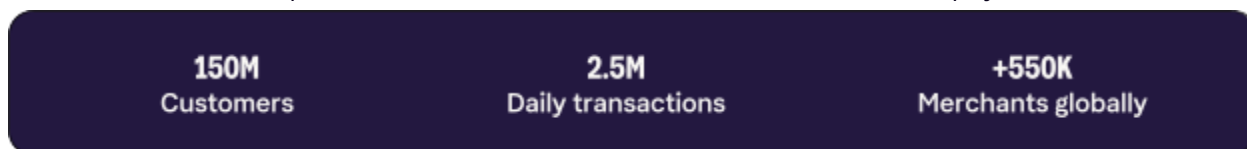
This document was last updated on Sep 16, 2024.

1.3 Definitions

The definitions outlined within the Klarna Network Rules also apply to this document. In many cases other terms will be defined directly in the text rather than using a glossary to ensure clarity and ease of reference, making it simpler for readers to understand the context without having to cross-reference multiple sections.

1.4 Understanding Klarna

Klarna has formed partnerships with a broad array of international Partners to make Klarna's flexible and convenient payment option the default checkout choice for customers worldwide. Each month, millions of customers opt for Klarna for their transactions, both online and in physical stores.



Partners grow their business with our flexible payment option and smart shopping solutions that enable customers to easily and securely pay when and how they want everywhere - online, apps and in physical stores. Klarna accommodates all shopping scenarios, from high-value transactions to everyday purchases and microtransactions. Partners using Klarna see:

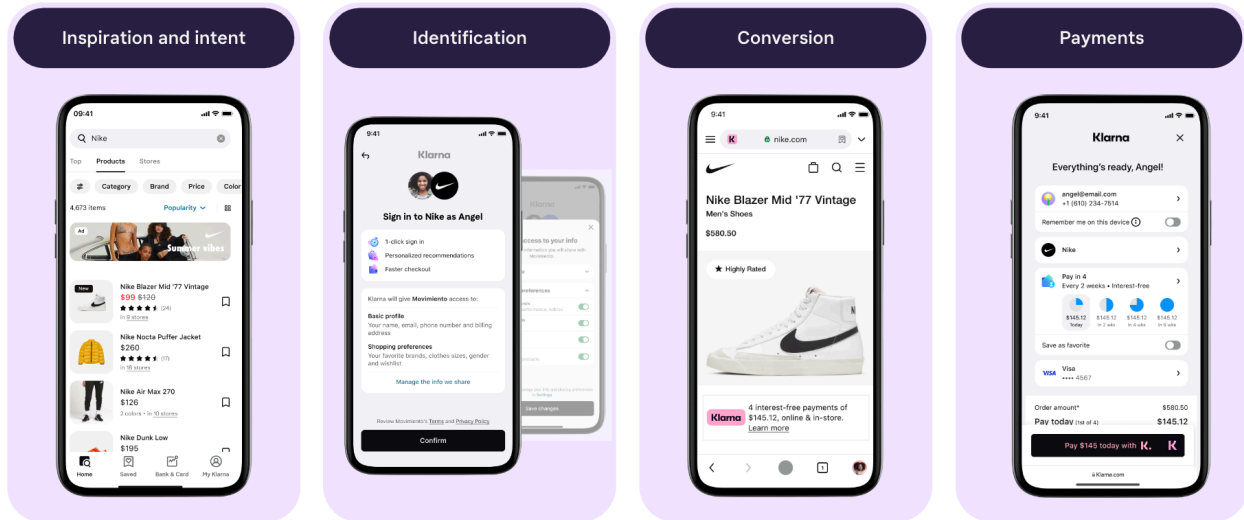
- **41%** Increase in average transaction value.
- **30%** Increase in conversion.



- **45%** Higher purchase frequency than average customers.

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

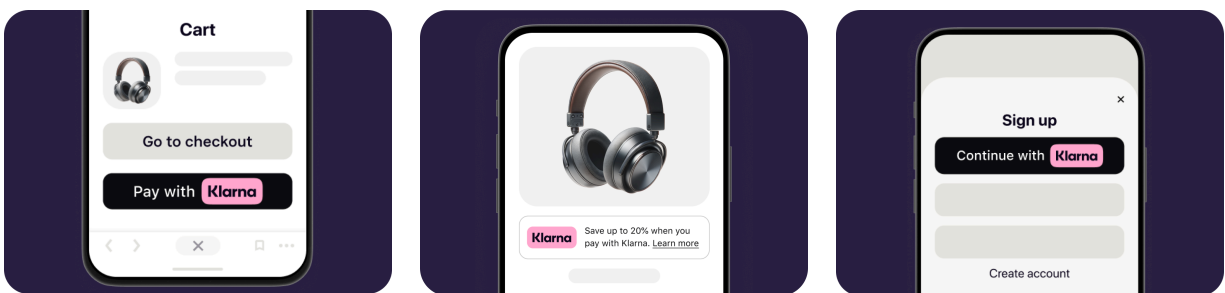
Our dynamic expertise ensures seamless interoperability between products within the Klarna ecosystem, delivering a smooth and consistent customer experience across all integration patterns.



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

1.4.1 Smart solutions to maximize sales

Our offering includes On-site messaging, Express checkout, and Sign in with Klarna alongside marketing services such as affiliation and price comparison search. These features are designed to improve customer experience and Partner sales. To ensure full product interoperability, Acquiring Partners are required to allow all Klarna services in the Product Suite to function seamlessly together following the recommendations in [Enable interoperability of Klarna product suite](#).



Klarna Express checkout

Offer a checkout process that is **6x faster**, significantly lowering the threshold for customers to complete a purchase.

On-site messaging

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

Sign in with Klarna

A **one-click experience**, increasing account registrations, (unrivaled **access to consented customer data**) and improving conversion rates.



2. Provide Klarna to my Partners

2.1 Before you start

Klarna offers a global and interoperable platform designed to support Partners management, transaction processing, post-purchase operations, and conversion rate optimization. Powered by the Web SDK and supported by extensive JavaScript SDK capabilities alongside Management and Partner Product APIs, this platform ensures seamless integration. As a Klarna partner, you are equipped to maximize the benefits of all available Klarna features.

As an Acquiring partner, understanding our solution principles and committing to them is essential before proceeding with integration and offering Klarna product suite to your Partners. This foundational understanding ensures that you can provide a best-in-class experience to your Partners, streamlining the integration and the adoption of future enhancements. A more detailed exploration of the following principles and their influence on the Klarna Network requirements, recommendations, and integration path is available in the Klarna Network Rules.

2.1.1 APIs overview

Management API	Global management of merchant lifecycle with Klarna, optimizing performance and unlocking ubiquity: Easy onboarding and end-to-end Partner Accounts management including: <ul style="list-style-type: none">• Global support by design• Onboarding and offboarding• Pricing management and querying price plans anytime• Integrated, automated, transparent fraud management enabling the ability to operate in real time.• Immediate automatic notifications via webhooks.• Flexibility of settlements and reconciliation process.
Partner product API	Easy access and scalable distribution of the Klarna Product Suite through: Single API suite to access all Klarna products and features, enabling: <ul style="list-style-type: none">• Payment API (Request, Token, Transaction)• Notifications API (Webhooks and signing keys)• Messaging API (Descriptors and Dynamic placements)• Shopping Session API• Settlements API• Disputes API Global harmonized endpoint Authentication for all features



2.1.2 Integration overview

These guidelines are divided into different sections covering the complete integration journey.

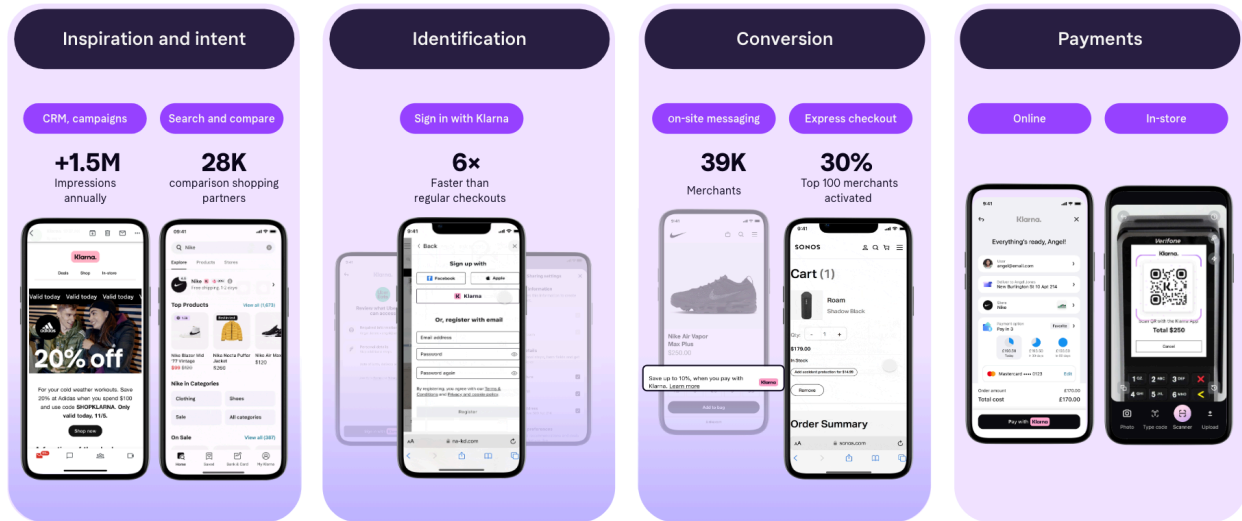
1. **Provide Klarna to my Partners**
 - a. [2.1 Before you start](#)
 - b. [2.2 Design your Klarna Solution](#)
 - c. [2.3 How to present Klarna in the checkout](#)
2. **Set up your partner account**
 - a. [2.4.1 Step 1: Configure account credentials](#)
 - b. [2.4.2 Step 2: Configure Klarna webhooks](#)
 - c. [2.4.3 Step 3: Account configuration management](#)
3. **Onboard and manage Partners**
 - a. [2.5.1 Step 1: Determine account structure](#)
 - b. [2.5.2 Step 2: Onboard Partners](#)
 - c. [2.5.3 Step 3: Manage your Partners payment products](#)
4. **Processing payments with Klarna Payment Services**
 - a. [2.6.1 Online store transaction](#)
 - b. [2.6.3 More payment use cases](#)
5. **Enable interoperability of Klarna product suite**
 - a. [2.7.1 Step 1: Grant access to Klarna's Partner portal](#)
 - b. [2.7.2 Step 2: Consume and pass key identifiers](#)
 - c. [2.7.3 Step 3: Consume Klarna interoperability data](#)
 - d. [2.7.4 Step 4: Consume Klarna Payment Confirmation Token and confirm Klarna payments](#)
 - e. [2.7.5 Step 5: Consume Klarna customer token](#)
6. **Managing Payment Transactions**
7. **Enable Post Purchase Operations: Facilitate seamless post-purchase activities.**
 - a. [2.9 Disputes handling](#)
 - b. [2.10 Pricing and reconciliation](#)
8. **Finalize your Klarna Integration**
 - a. [2.11 Test your integration](#)
 - b. [3.4 Create public documentation for your Partners](#)
9. **Klarna integration principles**



2.2 Design your Klarna Solution

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

In the following sections, you will find essential focus areas to consider when crafting your solution for partners.



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

2.2.1 Klarna product suite interoperability

To help Partners improve customer experience and maximize conversion rates, Klarna has developed a product framework that guarantees effortless interoperability of Klarna's Product Suite. This framework enables seamless functionality across systems, platforms and applications, not only streamlining Partners operations but also enhancing the overall shopping journey, and ensuring global consistency. We want every customer who visits your Partners' store to make a purchase.

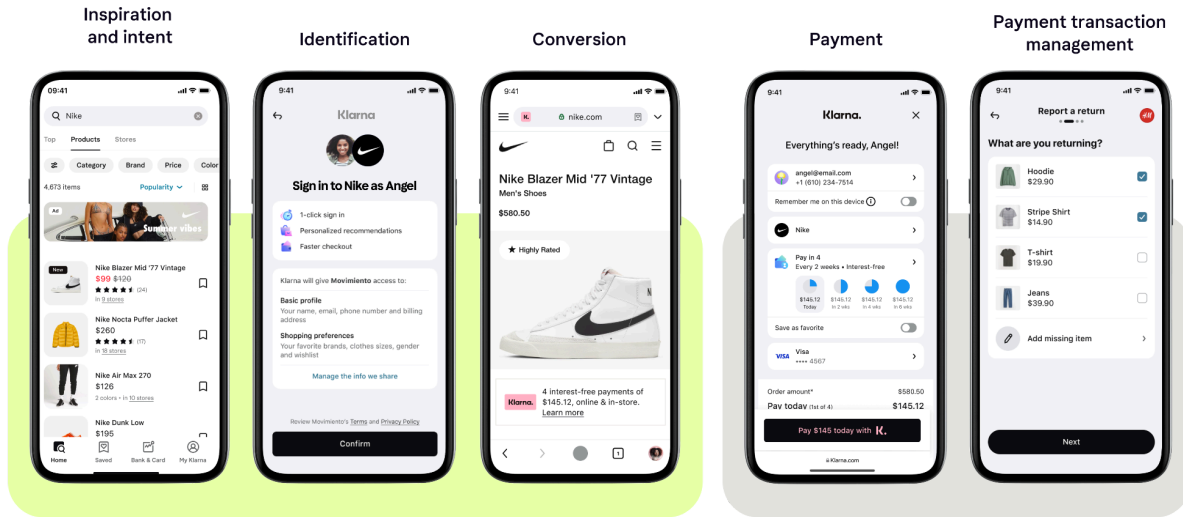
Partners have full access to the complete suite of Klarna products and services within this framework. By enabling the recommended integration flow outlined later in this document, Partners can achieve:

- **Flexibility:** Tailor the complete range of Klarna product suite to their unique needs.
- **Better conversion rates:** Leverage conversion-boosting Klarna features to reduce customer drop-offs and increase the likelihood of successful purchases, regardless of the integration approach for processing payments.
- **Higher customer satisfaction:** Provide a reliable and familiar payment experience that enhances trust and satisfaction, encouraging customer retention.



Integrated through Klarna APIs

Integrated through Acquiring partners APIs



Enabled by interoperability requirements

Enabled by payment integration

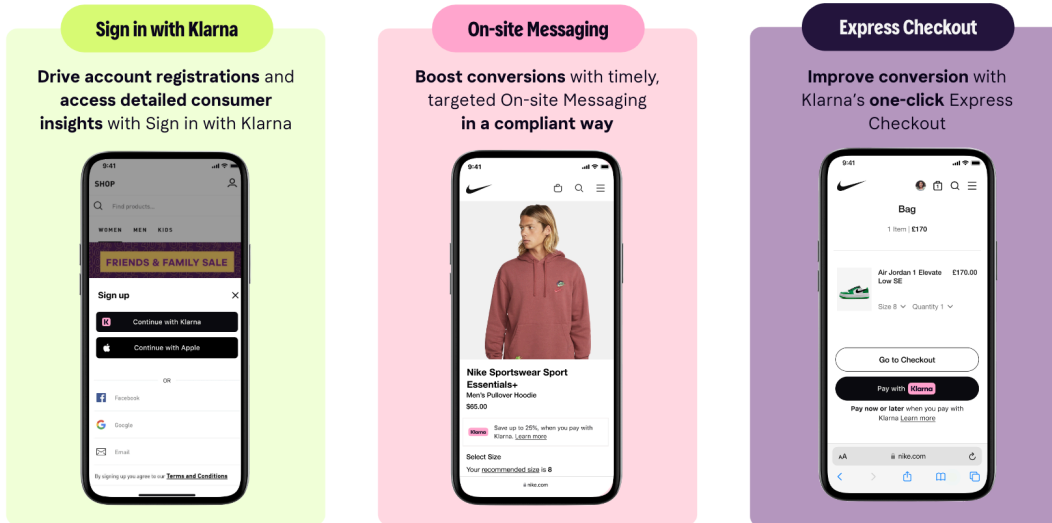
2.2.1.1 Accessing Klarna's Partner portal

The primary way to access the features available through interoperability is via Klarna's **Partner portal**. Access to Klarna's Partner Portal is made available via deep link from the Partner's dashboard. This deep link enables merchant Partners to click through and seamlessly gain access to the Partner portal via a temporary user account. Further details on this are provided in [Creating a deep link](#).

2.2.1.2 Product accessible via interoperability

Sign in with Klarna, **On-site messaging** and **Express checkout** are Klarna's conversion booster products. These features enhance the customer experience at various touchpoints, before and during the purchase process. Each feature targets a specific aspect of the shopping journey, and they deliver maximum impact when used together.

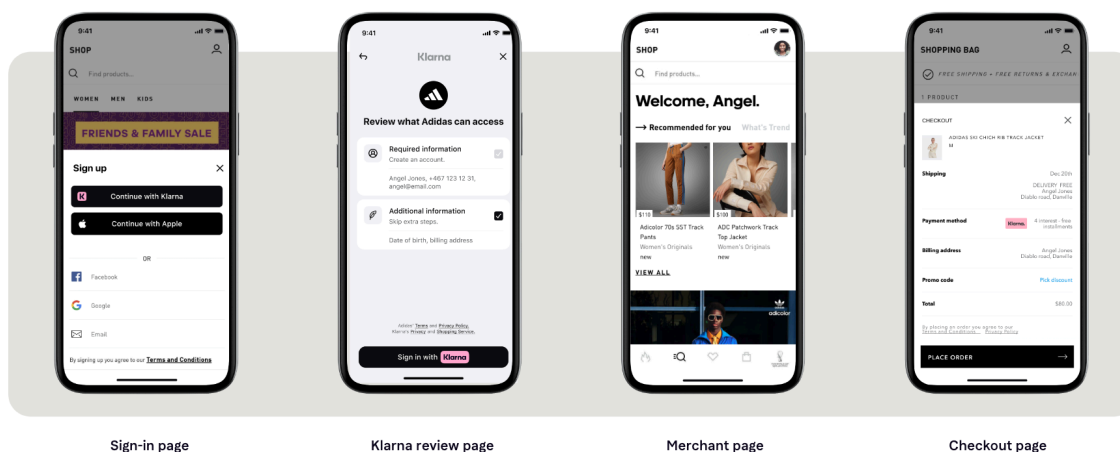




Tools to help Partners achieve maximum growth.

2.2.1.2.1 Sign in with Klarna

Partners strive to enhance the purchase experience for their customers. By leveraging Klarna's community of over 150 million customers, the Sign in with Klarna feature lets customers quickly and securely sign up on the Partner's website by using their Klarna account information. This allows Partners to identify their customers early in the shopping journey.



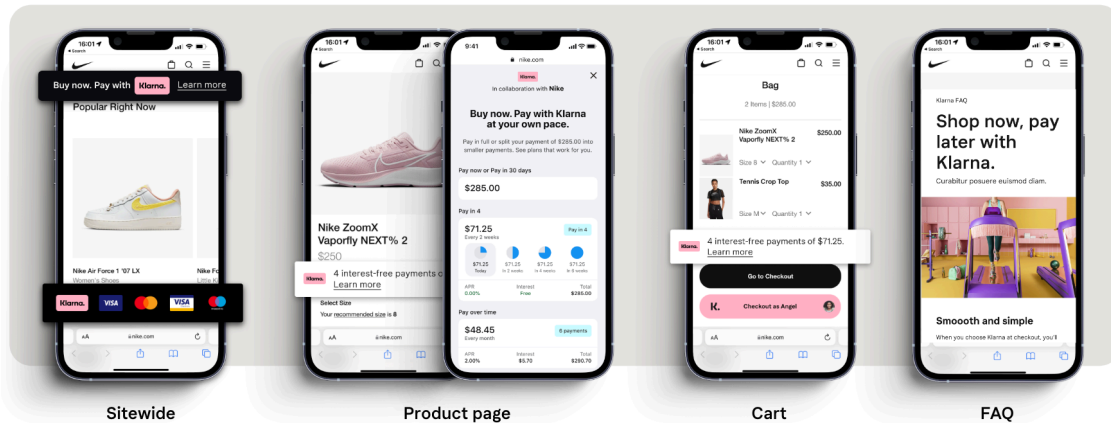
Easy registration, sign-in and checkout. Plus unrivaled access to detailed customer data.

When a customer registers with Klarna in an online store, the Partner gains additional intelligence that will enable the enhancement of the shopping journey, understanding the customer's needs and delivering a personalized experience.

2.2.1.2.2 On-site messaging

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





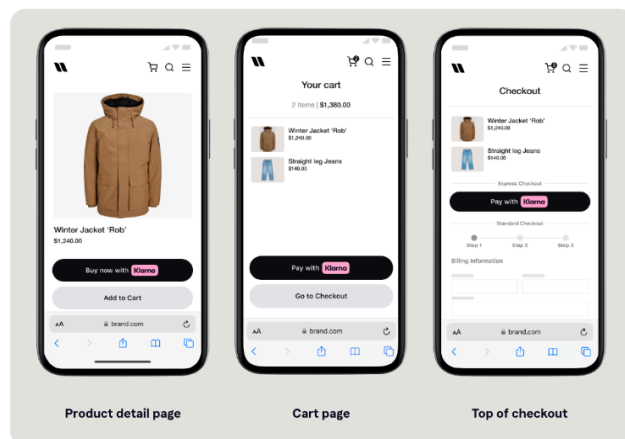
On-site messaging touchpoints: By adding On-site messaging to the shopping journey, Partners can inform customers about promotions, available payment methods and a payment calculator.

The look and feel for these dynamic placements is customizable and Partners are given the control to choose their preferred font, text style, size and logos. This flexibility allows them to match the aesthetics of their overall brand and website.

2.2.1.2.3 Express checkout

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

Partners can strategically place Klarna Express Checkout where customers are most likely to finalize their purchase. Displaying it early in the shopping journey allows customers the option to skip ahead when they're ready to buy.



More Information on Interoperability is available in [Interoperability](#).



2.3 How to present Klarna in the checkout

2.3.1 Checking Klarna availability

Klarna recommends that Klarna Partners verify the suitability of a payment and dynamically retrieve messaging during checkout using Klarna's Messaging API. By implementing this way, Klarna Partners ensure that the appropriate payment methods are always presented and no technical changes are required to expand with Klarna, simplifying scaling and ensuring a global partnership.

To support this dynamic capability, Klarna Messaging API enables you to dynamically display accurate payment descriptors based on your account settings and transaction specifics. This tool provides crucial information and visuals, ensuring global compliance and helping you effectively showcase Klarna-branded elements to enhance customer conversion rates.

2.3.2 Checkout structure

Familiarize yourself with the different components of the checkout page and how to display Klarna in the Partner payment selector. There are three main items to consider:

1 Payment descriptor

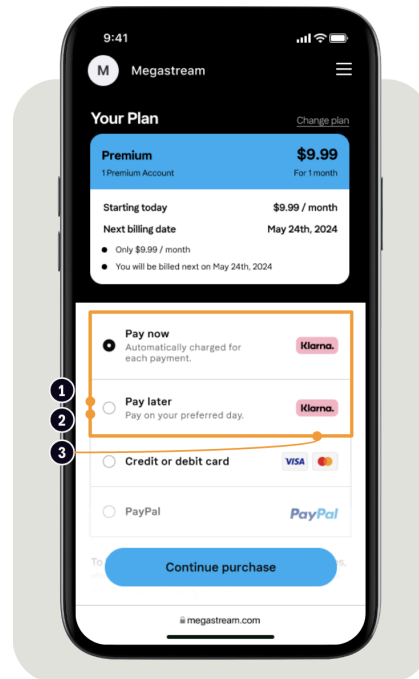
High-level call to action.
Provided in the response.

2 Payment subheader

More detailed breakdown of the value of the option presented. Provided in the response.

3 Klarna badge

Klarna logo. Provided in the response.



These may vary depending on the language and market. Klarna will provide these dynamically as part of the API response. More information is available in [Get content for Klarna's payment badge, descriptor, and subheaders](#).

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

2.3.2.1 Retrieval of Descriptors

For integrations capable of dynamically presenting payment methods based on Klarna's response, the recommended approach is to tailor your presentation of Klarna accordingly. This strategy optimizes conversion by providing clear information on available options and offers flexibility to

expand into additional countries as they become compatible, ensuring a future-proof and resilient integration. Moreover, this approach facilitates the presentation of promotional deals to the customer, enabling Klarna to help boost conversion rates in your checkout.

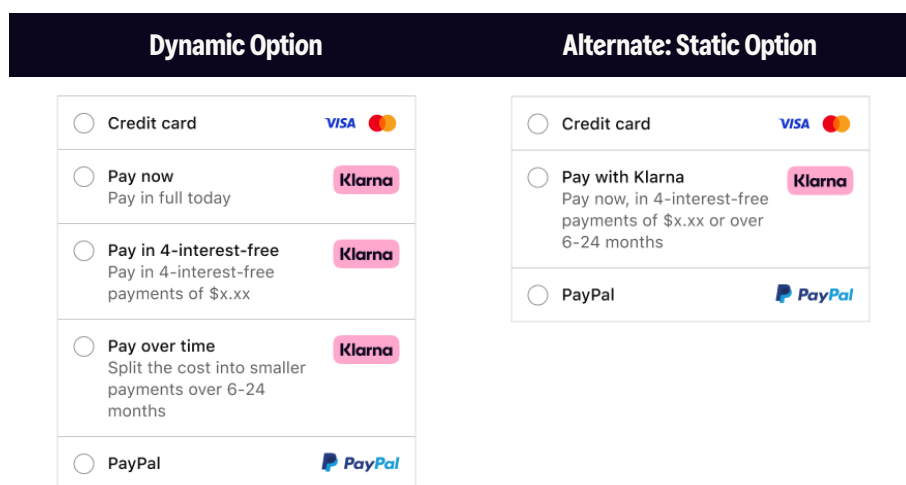
Consult [Check Klarna payment capability and display Klarna at the checkout](#) for more information on presenting Klarna dynamically in checkout.

2.3.2.2 Alternate flow: static presentation of Klarna

If your integration or that of the Partners 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 customers, maintaining consistency and simplicity.

When implementing a static presentation of Klarna, it is crucial to keep the names and details up-to-date and aligned with Klarna's best practices. You are responsible for maintaining this consistency. Regularly check and update the checkout copy to align with [Klarna's UX experience guide](#). Klarna recommends programmatically retrieving and caching the checkout language from Klarna, even if it can't be dynamically updated during checkout. These methods provide Partners with the flexibility needed for a globally adaptable checkout experience, aligning with their various customer experience requirements and expectations.

Consult [Display the Klarna checkout placement](#) for more information on presenting Klarna statically in checkout.



2.3.1.3 Pay with Klarna button

Allowing customers to complete their purchases using Klarna's JavaScript SDK simplifies the checkout process. By implementing the Partner product API package as detailed [here](#), you can easily display the payment button and handle the necessary actions when it's clicked. Once the payment button is clicked, Klarna takes over the payment process, allowing customers to complete their transactions efficiently.

Consult [Initiate payment request client-side using the Klarna payment button](#) for more information on presenting the Klarna payment button.



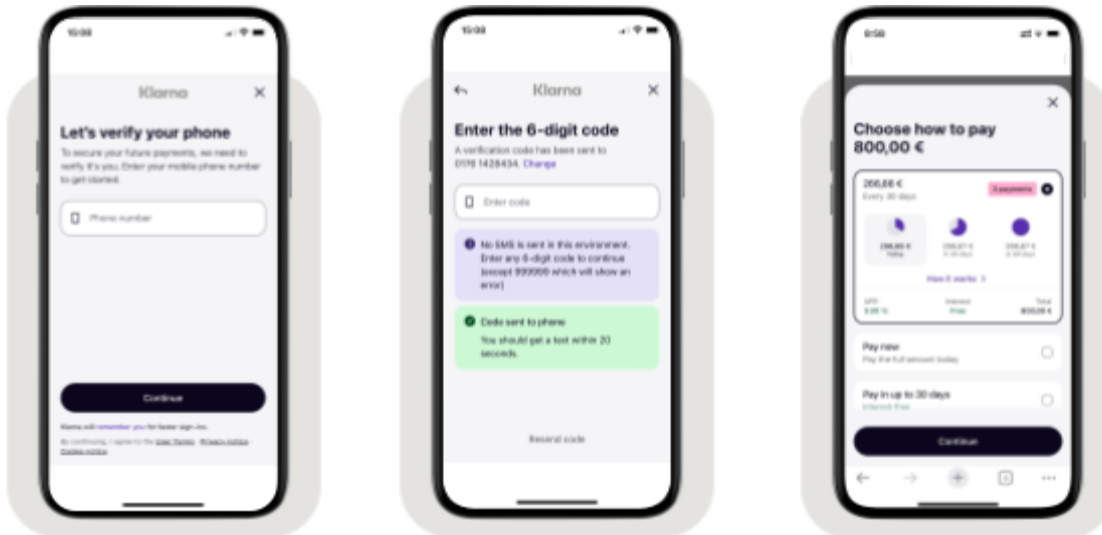
Pay with Klarna

2.3.3 Payment flow

Once the customer 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, which will be determined by the infrastructure of your payment solution:

- [Klarna.js integration](#)
- [Server-side only integration](#)

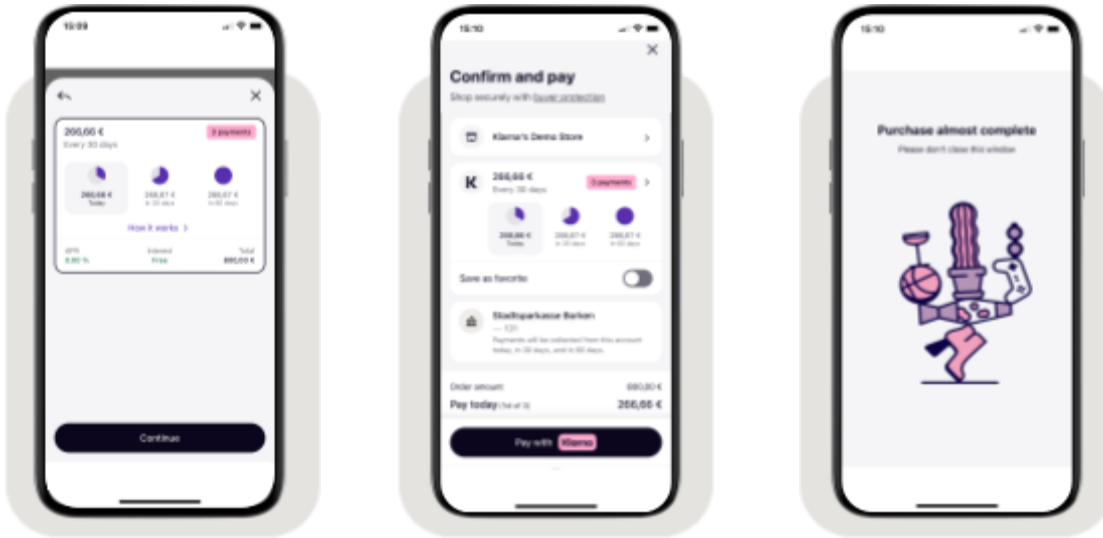
Klarna payment flow provides customers with a smart, consistent, and predictable experience, regardless of where they shop or how they want to pay. It offers a seamless process for both new and returning customers, as account creation is handled within Klarna's modal.



After verifying their phone number, new Klarna customers provide their email, billing address, and additional personal information. Klarna Partners can prefill most of these data points by initializing the payment request in the customer object.

Upon successful authentication, customers are invited to select one of the payment options offered by Klarna or to proceed with their previously selected preference.





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

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

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

Consult [Step 4: Monitor payment state and retrieve payment confirmation token](#) for more information on monitoring and progressing the payment flow.

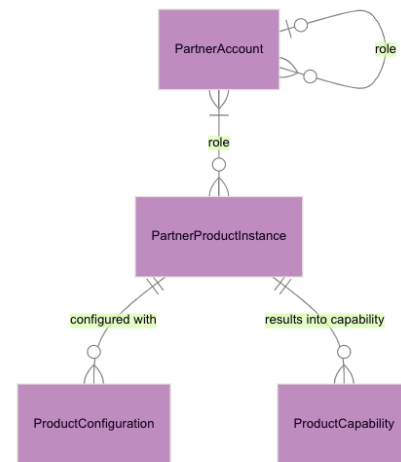


2.4 Set up your partner account

2.4.1 Partner account

A partner account is a construct that consolidates all relevant information, capabilities, and features based on agreements made with a Klarna partner. It acts as a central repository for various types of data related to a partner, analogous to a computer folder 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 may include roles for Partners and an acquiring partner, with each role having specific responsibilities.



2.4.1 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, you will receive your initial API key through a secure link.

With your initial API key from Klarna, you can create new API keys and Client IDs through the Partner Management API. Here's how these credentials function:

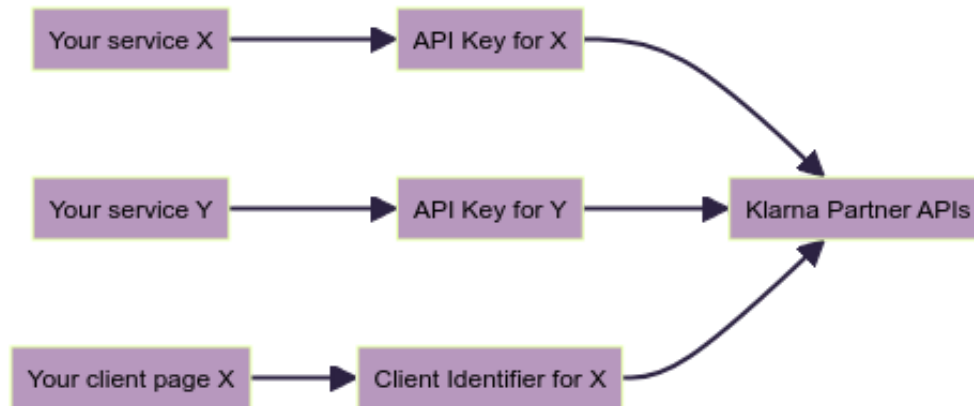
- **API keys:** Are used to authenticate server-side REST API requests towards Klarna. In addition, Klarna may use them to identify the source account.
 - Structure: `klarna_<live|test>_<api>_<random>`
- **Client IDs:** Are used to authenticate client-side interactions towards Klarna's SDK.
 - Structure: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

2.4.1.1 Account credential management

Credential management is fully under your control as an Acquiring partner, allowing you to create and manage credentials for different services. This autonomy enhances security by enabling automatic credential rotation, reducing the need for manual updating by Klarna.

To minimize risk in case of a security breach, **assign distinct credentials to each service**. If one credential is compromised or needs to be disabled, it will not impact the others, ensuring uninterrupted operation across your integration.



Other recommended practices:

- Never store credentials in plaintext.
- Implement strict access controls to limit credential exposure.
- Regularly rotate credentials.
- Immediately revoke and notify Klarna support if credentials are compromised.

Credentials can be managed for either live or test environments and are specific to client-side or server-side actions. When creating credentials, you can add a description to specify their use, which can be verified through a GET request to </v1/account/credentials>.

⚠ Credentials inactive for two months will be disabled to prevent misuse and deleted after ten months of inactivity. In such cases, you can reactivate old credentials or generate new ones through Partner support, your Klarna account representative, or via APIs, maintaining the security and flexibility of your interactions with Klarna.

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

1. Creating new credentials.
2. Transition your service from existing credentials to the new credentials.
3. Validate the new credentials have been correctly implemented before you
4. Make 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, and [Security](#) for more information about securely integrating Klarna.

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](#).

2.4.2 Step 2: Configure Klarna webhooks

Klarna webhooks enable your applications to receive real-time business event notifications from the integrated Klarna product suite, allowing your backend systems to respond promptly.

Webhooks are customizable, and all Klarna APIs include a standard set of webhook events to which you can subscribe. For a list of supported event types, refer to the [Events categories](#) section.

Consult the [API reference](#) for a complete description of the request body parameters.

2.4.2.1 Getting started with Klarna webhooks

To maintain seamless integration with Klarna's payment services and ensure your systems are equipped to handle Klarna notifications effectively, follow these steps:

1. Establish a secure endpoint

- a. Expose an HTTPS endpoint on your server designed to receive webhook notifications.
- b. Ensure only HTTPS endpoints are used and that a valid SSL certificate is in place.

2. Endpoint Configuration

- a. Configure a single endpoint for multiple event types or set up separate endpoints for each event type based on your preferences.
- 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.
- c. Store the webhook event data for further processing.

Example 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",
    "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}}
```

4. Acknowledge the webhook

- a. Immediately respond to webhook notifications with an HTTP status code of 200, 201, 202, or 204 to confirm successful delivery.
- b. Failure to respond, or a response with any other status code, will trigger Klarna's retry mechanism.

5. Retry mechanism

Klarna will retry notifications at increasing intervals upon failure (timeout or HTTP 4XX/5XX response codes). 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:

- First retry: 10 seconds after the initial failure.



- Second retry: 2 minutes later.
- Third Retry: 15 minutes later.
- 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.

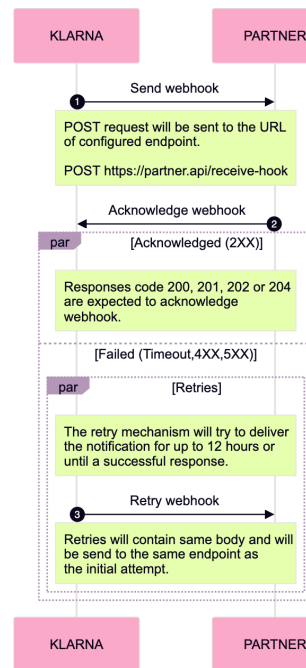
Handling failed notifications

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.



⚠ 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.

2.4.2.2 Create and manage signing keys

To securely receive webhook notifications, you must generate and attach signing keys to your webhooks. This procedure ensures the authenticity of incoming notifications can be verified. Klarna will sign each notification using the designated signing key before dispatch. Both the signature and

its identifier will be included in the notification, enabling you to confirm its validity. When configuring a new webhook, specify the signing key to be used.

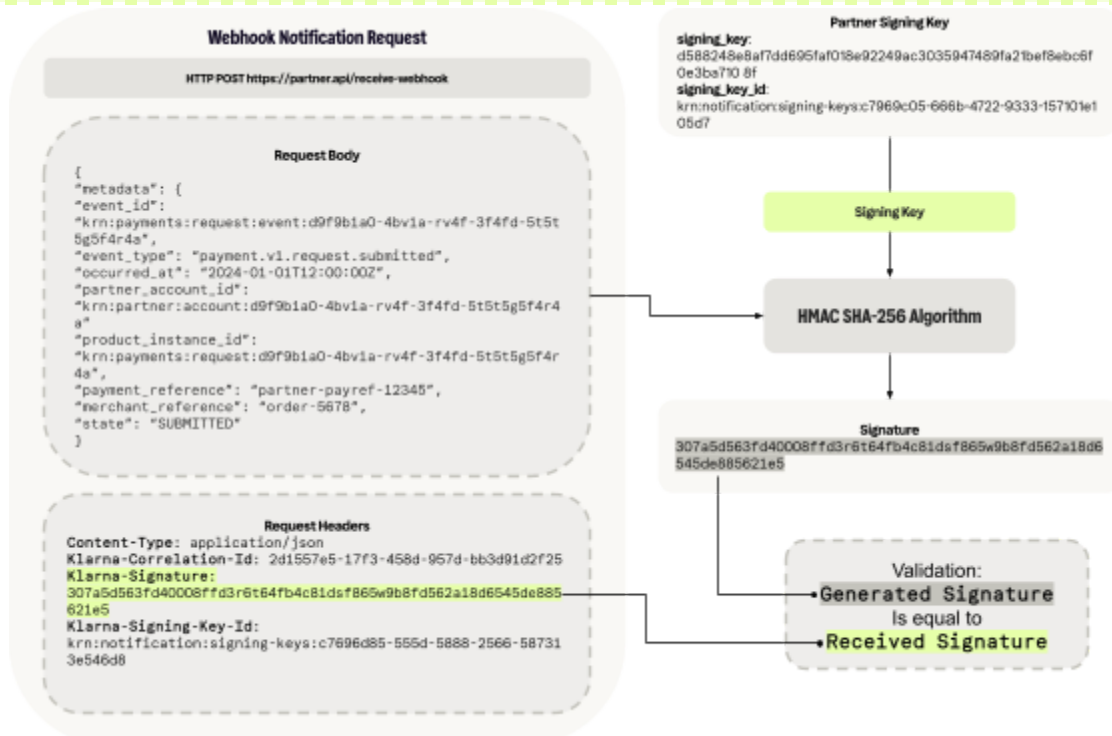
⚠ Important: After generating signing keys, make sure to securely store them immediately. You will not be able to retrieve these keys again for webhook validation purposes.

2.4.2.2.1 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.

Release notes

17 Currently a maximum of 50 signing keys per account are supported. If new keys are required, a DELETE request to /v1/notification/signing-keys/{signing_key_id} must be performed to disable the old keys.



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:

1. Creating a new signing key.
2. Updating the recipient server to accept the new signing key.
3. Updating the webhook.
4. Validate the new signing key has been correctly implemented before
5. Remove the old signing key.

Consult the [API reference](#) for a complete description of the request body parameters.

2.4.2.3 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

2.4.2.3.1 Events categories

Explore the categories of events available through Klarna webhooks, each designed to keep you informed about specific aspects of your transactions and account activities:

- **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 Partner'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 your webhooks are set up, you can manage them through API requests. Use these requests to update, delete, or retrieve details about your configurations. For comprehensive parameter information, consult the [API reference](#).

Specific webhooks related to individual components of your Klarna integration are detailed in the integration guidelines for those products. Below are the key integration areas where webhooks play a vital role:

- [Account lifecycle and management webhooks](#)



- [Client-side payment request notification events](#)
- [Server-side payment request notification events](#)
- [Transactional Dispute webhooks](#)
- [Settlement webhooks](#)

2.4.2.4 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 and is critical for ensuring your integration functions as expected before going live. By manually triggering webhooks, you can simulate any state change for a payment request instantly, bypassing the need to complete the purchase flow or wait for the request to expire. This testing capability allows you to validate the creation, confirmation, and state changes of payment transactions. Additionally, it enables you to receive notifications for expired payment requests, which can be useful for abandoned cart strategies or similar retargeting purposes.

In the test environment, when you trigger a simulated webhook, the endpoint generates a test event using dummy data that adheres to the schema of the specified event type. This approach ensures you can thoroughly test and adjust your system's response to various webhook events without affecting live data.

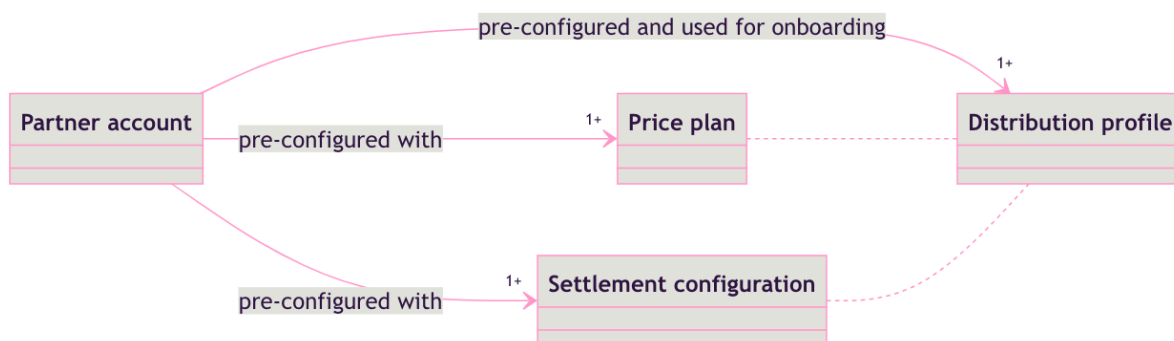
Example webhook event:

```
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",
    "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"
  }
}
```

2.4.3 Step 3: Account configuration management

This section details the available configurations for setting up accounts and offers recommendations on their utilization.

As a partner integrating and distributing Klarna solutions, we collaborate closely during the account setup process. Your Partner account will be granted access to specific resources assigned by Klarna, including price plans and settlement configurations. These tools are designed to optimize your operations and ensure a seamless integration with Klarna services.



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

17 Further configuration points may become available with future releases. Distribution Profiles will be deprecated with release 4.

2.4.3.1 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 encompasses rates that can vary depending on the market, payment program, and channel type. Additionally, microtransaction caps may override the base pricing. Klarna creates and maintains these price plans.

Release notes

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

2.4.3.2 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 Partner.
- **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 acquiring partner integrating Partners through Klarna, you manage their settlements, thereby simplifying their reconciliation process. 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. Local-Pay 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 Partners, regardless of their onboarding country or entity. Therefore, Local-Pay does not need to specify any specific settlement configurations during the Partner onboarding process.
2. In contrast, Global-Pay 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. Global-Pay prefers payouts for Partners onboarded in the UK to be directed to BankAccount1 for all currencies, while
 - b. Payouts for Partners onboarded in the EU should be directed to BankAccount2 for all currencies.

Consequently, Global-Pay needs to specify the applicable settlement configurations during the Partner onboarding process.



2.5 Onboard and manage Partners

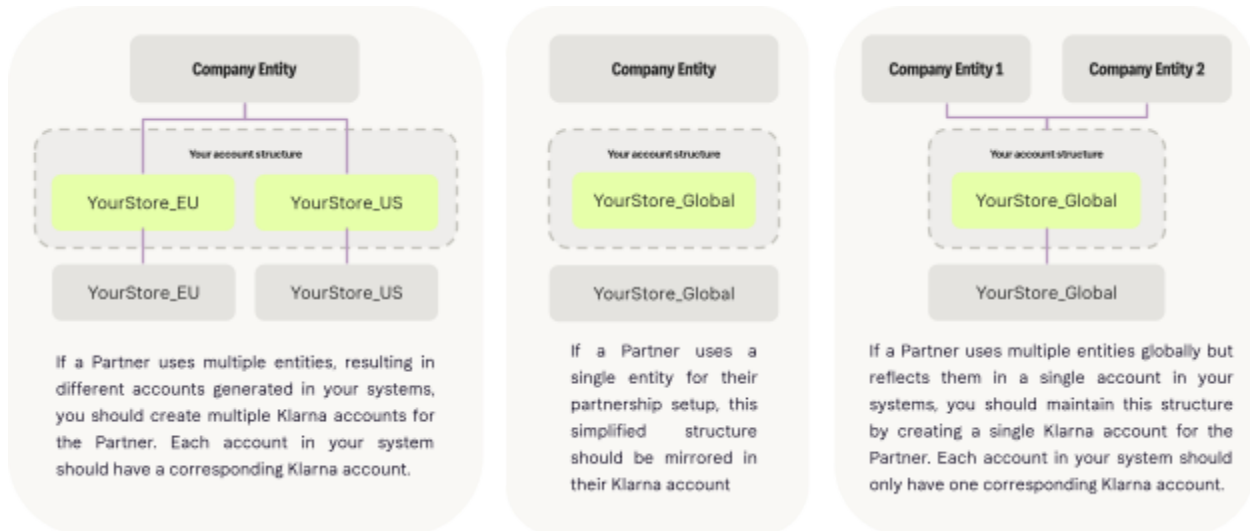
2.5.1 Step 1: Determine account structure

Partner accounts are used for managing the relationship between your Partners and Klarna Payment Services. These accounts, created and managed via Klarna's Management API, contain critical information to facilitate a successful partnership. These blocks consist of:

- **Account owner:** Information about the main representative of the Partner 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 Partner operates through multiple entities, we prefer that the account structure be simplified towards Klarna by mapping to a single account only, however you as the Acquiring partner **should align the creation of the Klarna Partner account to the structure reflected in your own systems.**

Overall, it is required that a Partner should require no additional effort to onboard Klarna as compared to cards or any other low-friction payment method.



Release notes

⚠️ Current Limitation: Each account can currently support only one website channel, which restricts support for multi-website or physical stores. Further details on channel management can be found in the [Channel types and management](#) section.

📅 17 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 Partners lifecycles are available in the [Manage your Partner](#) section.

2.5.1.1 Account structure use cases

Let's take a look into a few different examples of how Partner accounts can be used to represent Partners:

2.1.1.1 Retail

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

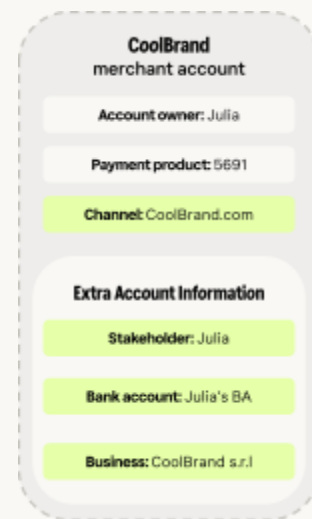
Example 1:

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 Partner.



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-337153c04a58",
      "type": "PAYMENT",
      "merchant_category_code": "5691"
    }
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://CoolBrand.com"
        ]
      }
    ]
  }
}
```

2.5.1.1.2 Multiple MCCs

In case a Partner has multiple MCCs and operates under a single account, there are two different ways that the model can be applied. These approaches can be applied to Partners that sell different types of products or marketplaces.


2.5.1.1.2.1 Single MCCs provided at onboarding

Release notes

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

The first way would be the same as previously indicated, where **only the primary MCC is set during onboarding**. Different MCCs can later be passed at transaction time, and override the primary MCC.

This approach should be used when you don't have the list of MCCs the partner operates at during onboarding. Utilizing a too-different or restrictive MCC at the time of transaction might lead to the transaction being rejected.

 If the Partner is operating in restricted verticals, the restricted MCC must be used for the onboarding of the Partner. On a transactional level this can be overridden by the unrestricted MCCs applicable to the account.

2.5.1.1.2.2 Multiple MCCs provided at onboarding

Release notes

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

If you have the list of MCCs the partner is using at the point of onboarding, those MCCs can be passed on the *enforced MCCs* list as part of the payment product. When the enforced MCC list is present, providing an MCC that is not the list when initiating a transaction may result in the transaction being rejected.

Example 2.1:

Consider a Partner called "AstroxSox". This business sells new socks to its customers, but also provides a subscription service so their customers never run out of socks. This means that AstroxSox operates both under MCC 5691, but also 5641.

AstroxSox is owned by Mathias and his sister, Debora, and operates under a single website, astroxsox.de.

In this case, the Partner must indicate the appropriate MCC at time of transaction, allowing correct pricing and payment options to be shown. The specific MCC is passed when initiating a payment request.

AstroxSox
merchant account

Account owner: Debora

Payment product: 5691

Channel: AstroSox.de

Extra Account Information

Stakeholder: Mathias

Stakeholder: Debora

Bank account: AstroxSox's BA

Business: AstroxSox GmbH

JavaScript

```
{
  "account_reference": "M12378999999",
  "account_name": "AstroxSox",
  "account_owner": {
    "given_name": "Mathias",
    "family_name": "Doe",
    "email": "Mathias@AstroxSox.de",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
      "krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a58",
      "type": "PAYMENT",
      "default_merchant_category_code": "5691",
      "enforced_merchant_category_codes": ["5641"]
    }
  ],
  "channel": {
    "websites": [
      {
```

```

    "urls": [
      "https://AstroSox.de"
    ]
  }
]
}}

```

Example 2.2:

If "AstroSox" is set up in such a way that the partner has configured two accounts in their system.

In this case, it can be modeled the same way on Klarna's side by creating two separate accounts as illustrated below. In this instance, AstroSox would leverage the appropriate account for a given transaction, and would be required to keep all account-specific tasks siloed. Handling disputes, payment transaction management, and settlements separately for each account.



Which would require two onboard calls, first:

```

JavaScript
{
  "account_reference": "M12378999999-1",
  "account_name": "AstroSox",
  "account_owner": {
    "given_name": "Mathias",
    "family_name": "Doe",
    "email": "Mathias@AstroSox.de",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-p
rofile:206bbb83-9b6e-46fa-940d-337153c04a5
8",
      "type": "PAYMENT",
      "merchant_category_code": "5691"
    }
  ],
}

```

And second:

```

JavaScript
{
  "account_reference": "M12378999999-1",
  "account_name": "AstroSox",
  "account_owner": {
    "given_name": "Mathias",
    "family_name": "Doe",
    "email": "Mathias@AstroSox.de",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-p
rofile:206bbb83-9b6e-46fa-940d-337153c04a5
8",
      "type": "PAYMENT",
      "merchant_category_code": "5641"
    }
  ],
}

```

```

],
"channel": {
  "websites": [
    {
      "urls": [
        "https://AstroSox.de"
      ]
    }
  ]
}
] }}

```

```

],
"channel": {
  "websites": [
    {
      "urls": [
        "https://AstroSox.de"
      ]
    }
  ]
}
] }}

```

2.5.1.1.3 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.

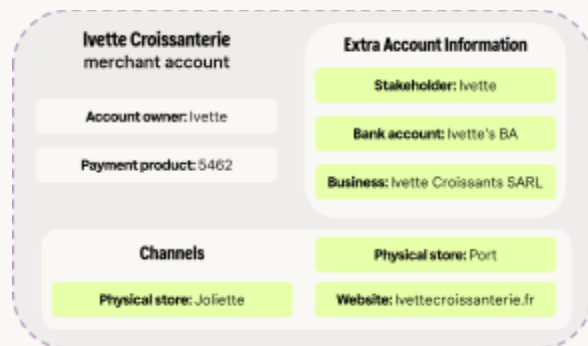
Example 3

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 ProductAPI.



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-337153c04a58",
      "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.

2.5.1.1.4 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, for some franchises a single account will suffice. This is dependent on how they wish to receive settlements.

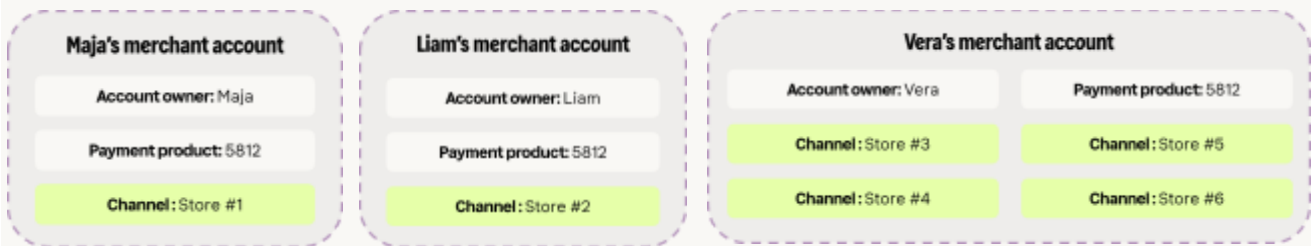


Example 4

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 (store 1), Liam (store 2) and Vera. Out of the three, Vera was able to run a really successful business and now owns four different burger locations, stores 3, 4, 5 and 6.

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 from which specific store the payment request originates. Applied to our example, Maja, Liam, and Vera would each have a separate account. Maja and Liam would have a single channel each, and Vera would have four channels, one for each of her physical stores. By passing the correct account ID and channel ID in the request, Klarna can present to the customer different store details and an experience personalized to the originating store.



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": "+4955555555"
  },
  "products": [
    {
      "distribution_profile_id":
      "krn:partner:global:account:distribution-
      profile:206bbb83-9b6e-46fa-940d-337153c04
      a58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    }
  ],
}
```

2nd:

```
JavaScript
{
  "account_reference": "M123789922223",
  "account_name": "Liam WTB",
  "account_owner": {
    "given_name": "Liam",
    "family_name": "Doe",
    "email": "liam.franchisee@wtb.se",
    "phone": "+4955555555"
  },
  "products": [
    {
      "distribution_profile_id":
      "krn:partner:global:account:distribution-
      profile:206bbb83-9b6e-46fa-940d-337153c04
      a58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    }
  ],
}
```

```

"channel": {
  "physical-stores": [
    {
      "name": "Maja WTB",
    },
  ]
}

```

```

"channel": {
  "physical-stores": [
    {
      "name": "Liam WTB",
    },
  ]
}

```

3rd:

JavaScript

```

{
  "account_reference": "M123789922224",
  "account_name": "Vera WTB",
  "account_owner": {
    "given_name": "Vera",
    "family_name": "Doe",
    "email": "vera.franchisee@wtb.se",
    "phone": "+4955555555"
  },
  "products": [
    {
      "distribution_profile_id":
"krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a58",
      "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.

2.5.2 Step 2: Onboard Partners

To start offering Klarna Payment Services to your Partners, you must first onboard them to Klarna.



During this process, it's crucial to define the appropriate structure based on the partner's business model, as detailed in the in [Determine account structure](#) section.

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

While only a subset of data points is mandatory from an API perspective to complete this phase, it is essential to provide all relevant and available information as specified in your data transfer and cooperation agreement with Klarna. Correct and accurately shared data allows Klarna to minimize potential risks and ensure a better experience to all parties.

When a Partner is onboarded with Klarna, Klarna will create an account for the Partner 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 Partner as it is a required parameter for the integration of the Partner product API.

The example provided here is for illustrative purposes only. The parameters required for onboarding may vary based on your commercial agreement and the parameters available to you. Please 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:test:LYIPRM59",
  "account_name": "John Doe Stakehouse",
  "account_reference": "M123786123412",
  "state": "PARTIALLY_OPERATIONAL",
  "state_reason": "INITIAL_SETUP"
}
```

2.5.2.1 Handling rejected onboardings

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

2.5.2.1.1 Identification of incomplete or incorrect data

Klarna's Partner management 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.

2.5.3 Step 3: Manage Partner payment products

Using the Management API, you have full control over Partner 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. Familiarize yourself with these processes and actively use the available tools to optimize account actions and product handling. The lifecycle diagram below provides detailed insight into these stages.



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 actions, such as post-purchase calls, may still succeed.
TERMINATED	The payment product is fully disabled by Klarna for AML, fraud or risk reasons.


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

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. More information on this process can be found in the [Account lifecycle](#) section.

2.5.3.1 Channel types and management

A key component of a partner account is the channel. Channels represent where Klarna Payment Services are going to be available to customers via the Partner. 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 customer 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 when making a payment request. This ensures the

correct brand identity is displayed to customers, enhancing their shopping experience. Accurate identifying the payment channel is also crucial for the effectiveness of Klarna's fraud assessment systems.

2.5.3.1.1 Channel collections

Channel collections are designed to unify brand identity across various customer touch points. Each channel collection includes Partner-specific details such as branding, support channels, and social media links, crucial for enhancing the customer's purchase and post-purchase experience, including interactions 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 manage branding efficiently across multiple channels:

1. Define the branding, support, and social media details within a channel collection.
2. Assign a unique `collection_reference` to this collection.
3. Apply this reference to each desired channel to ensure consistent branding without duplicating details for each channel.

Consult the [API reference](#) for a complete description of the request body parameters.

2.5.3.2 Keeping partner accounts updated

Acquiring Partners are required to use the Management API to properly update and maintain Partner data in Klarna's systems. We offer RESTful PATCH operations for all objects that can be updated, and the updates can be done as soon as they happen on your side.

Data integrity is essential

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

2.5.3.2.1 Disabling a product

Klarna relies on Acquiring Partners to safeguard the reliability and security of the Klarna Network. If a Partner account has only one product configured and it is disabled, the account will be unable to process transactions, but may continue to manage completed payments.

In case the Partner is suspended by you or they decide to close their account, you must immediately disable all products configured on the account so Klarna has an accurate understanding of the Partner status. Updates to the product configuration should reflect changes applied in your systems, and should not be triggered more than once per account per second where possible. Any update or disabling of a product may take up to 10 minutes to be reflected towards customers.

Depending on the status of a Partner when their Klarna products are disabled, Klarna may continue to receive outreach from customers for months after the disablement action. Ensuring accurate and

up to date account and product status towards Klarna will allow higher accuracy on adjudication of customer claims and other outreaches.

To indicate a product has been disabled within your system to Klarna the /disable endpoint may be called, indicating the reason why the payment product is being disabled.

Release Notes

17 Additional enums will be added in future releases.

Reason	Definition
FRAUD	In case Klarna Product is being disabled due to fraud suspicion. More information on fraud and account security is available in Payments on restricted businesses .
VOLUNTARY	In case the Klarna Product is being disabled due to voluntary cancellation of the account.

An open text details field is also available for more information about why the payment product is being disabled.

2.5.3.2 Re-enabling a product

In case a disabled payment needs to be re-enabled, be it because you identified that the Partner was not actually fraudulent or because the Partner decided to re-enable their integration, the /enabled endpoint can be used to achieve this.

2.5.3.3 Account lifecycle webhooks

A Partner account has a defined lifecycle, which **can only be manipulated by Klarna**. This lifecycle is intricately linked with the product lifecycle, and most processing logic should be developed around the product lifecycle.



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.
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

Parameter	Definition
	added to it. All products within an account are fully disabled as a part of this status.

It is essential to configure [management webhooks](#) to receive notifications about status changes in Partner accounts. Below is a table describing the events supported by Klarna webhooks for account management, which allows immediate notification and action on certain events.

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",
    "event_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"
  }
}
```



2.6 Processing payments with Klarna Payment Services

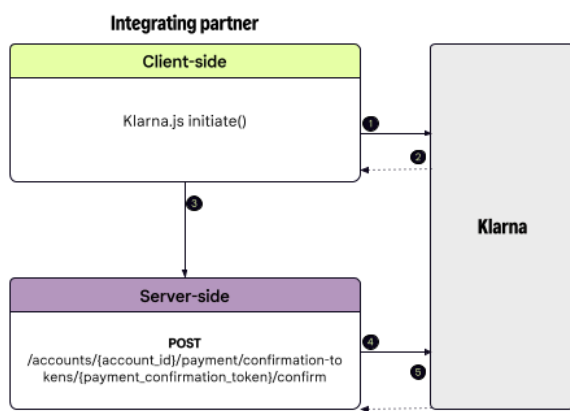
To integrate Klarna Payment Services as an Acquiring Partner, we provide flexible integration options to accommodate various integration styles. These options allow you to tailor the integration to your specific needs and the needs of your Partners.

For offerings in which the Acquiring Partner has a client-side component, such as hosted checkout solutions or embedded components, it's required to utilize a client-side integration leveraging Klarna's SDK, Klarna.js, and Klarna Partner product API. This ensures a seamless implementation of Klarna Payment Services directly on your platform, providing advanced functionality for security and ease of checkout.

Alternate server-side only integration.

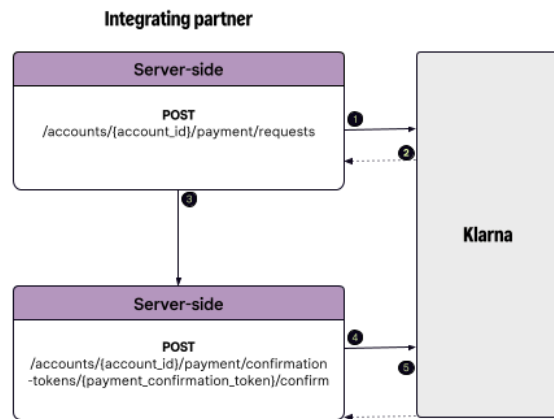
For offerings that solely provide server-side integration to Partners, where the Partner owns the client-side integration of Klarna, you should adopt the Partner product API and expose redirect links to your Partners. This ensures compliance while allowing for flexibility to accommodate your integration style. More information in [Server-side initiated payment request](#).

Client-side via Klarna.js and Server-side via Partner Products API



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.

Full server-side via Partner Products API



1. Initiate the payment request server-side and redirect the customer 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 `klarna's interoperability_token` to create a unified and seamless shopping experience, especially when your integration includes both client-side and server-side elements. This identifier is provided by the Klarna web SDK in response to any initiating action where an existing `interoperability_token` is not provided. By providing this parameter to the Partners



(where the initiation of Web SDK is owned by you), and allowing the Partner to pass the parameter in all your requests, you tie together these elements, ensuring consistent data flow and maintaining payment request continuity throughout the customer's journey.

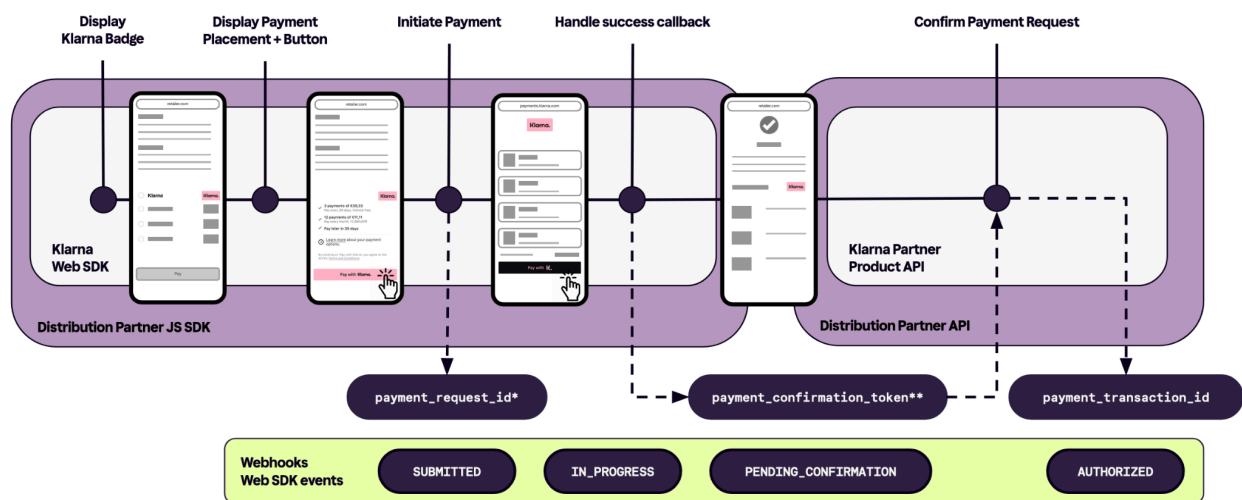
Implementing the klarna's interoperability_token enhances personalization, security, and cohesion in the checkout process, reducing friction and leading to higher customer satisfaction and increased conversion rates. More information on leveraging klarna's interoperability_token to improve the customer experience is detailed in the [interoperability](#) section.

2.6.1 Online store transaction

2.6.1.1 Payment request initiated via Klarna.js

A client-side integration via the Javascript library **Klarna.js** provides customers with a seamless checkout experience allowing them to complete their purchase without leaving the Partner's website. Furthermore, it also supports 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:



* Payment request expires after 48hr
 ** Confirmation token expires after 60m

2.6.1.1.2 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 initiated, Klarna.js provides an interoperability_token, which should be handled as outlined in the [Klarna interoperability token](#) section. Within your create payment request APIs, allow the Partner to provide an interoperability_token in case they have previously started a Klarna payment request regarding the customer arriving at the checkout.

Initialization of the Web SDK



```

JavaScript
<script type="module">
const { KlarnaSDK } = await import("https://js.klarna.com/web-sdk/v1/klarna.mjs")

const Klarna = await KlarnaSDK({
  "clientId": "[client-id]",
  "accountId": "[account-id]"
})

</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.

2.6.1.1.3 Step 2: Check Klarna interoperability status and display Klarna at the checkout

To integrate Klarna Payment Services seamlessly as an Acquiring Partner, we require that a structured approach is followed. Central to this is confirming the availability of Klarna Payment Services and customizing the payment messaging by using the `capability()` and `getPlacementContent()` methods. Doing this helps achieve an optimal checkout process, leading to increased customer satisfaction and higher conversion rates.

2.6.1.1.3.1 - Confirm availability and select Klarna as a primary payment method

Invoke the `resolve()` 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 Klarna's availability. The method also returns the state of the shopping session registered on the customer device. Select Klarna as a primary payment method if the state is `PAYMENT_PREPARED` or `PAYMENT_PENDING_CONFIRMATION`.

```

JavaScript
// interoperabilityToken is required when Klarna.js is loaded
// on your own domain, different from the merchant domain
const interoperability = await Klarna.Interoperability.resolve({
  interoperabilityToken: "[received_interoperability_token]"
}, {
  country: "SE",
  currency: "EUR",
  paymentAmount: 10000
})

switch (interoperability.status) {
  case "PAYMENT_PENDING_CONFIRMATION":
    // Confirm Klarna payment immediately, otherwise,
    // display Klarna as a selected payment method
    break;
  case "PAYMENT_PREPARED":
    // Display Klarna as a selected payment method
    break;
  case "PAYMENT_UNSUPPORTED":

```

```

// Don't show a Klarna button or offer Klarna
break;
case "PAYMENT_SUPPORTED":
default:
// Display Klarna as a payment method
break;
}

```

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

1 Payment descriptor

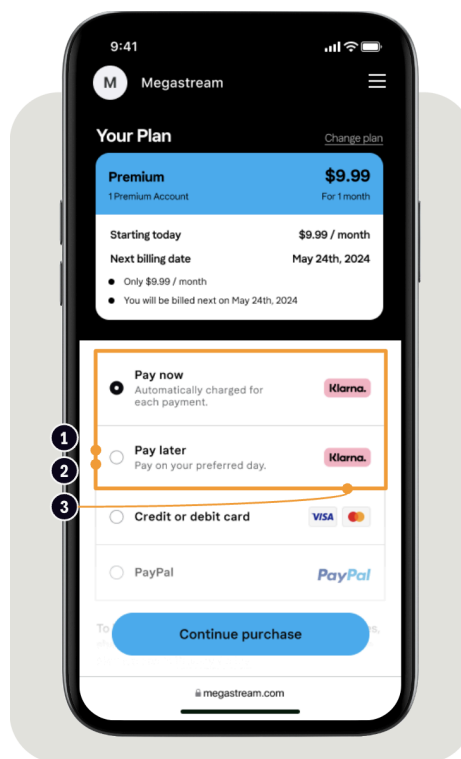
High-level call to action. Provided in Klarna response.

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.



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

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

To achieve a flexible and global solution that allows Partners to choose their preferred presentation method, as per [How to present Klarna in checkout](#), we require you to **allow your Partners to pass the category_preference parameter at checkout**. If no or invalid preferences are provided by your Partner, the default behavior is to display all available options separately.

Additionally, allow the passing of the `interoperability_token` at this point to tie the payment request to any existing client-side sessions which may have been initiated by the Partner.

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 `category_preference=KLARNA`) or
- Presented as a static payment method

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

Parameters

Parameter	Definition
<code>locale</code>	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
<code>currency</code> (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.
<code>payment_amount</code>	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.
<code>category_preference</code> (optional)	Indicates the preferred payment option. Enum: <ul style="list-style-type: none"> • PAY_NOW • PAY_LATER • PAY_OVER_TIME • KLARNA
<code>message_preference</code> (optional)	Indicates the use case of checkout. Only necessary for tokenized payments. Enum: <ul style="list-style-type: none"> • ECOMMERCE • SUBSCRIPTION • ON_DEMAND • IN_STORE
<code>subscription_interval</code> (optional)	Indicates the cadence of the subscription if <code>message_preference=SUBSCRIPTION</code> , i.e. "1-MONTH"

Consult the [SDK reference](#) for a complete description of the request body parameters.

For more detailed information on how to leverage the parameters specific to tokenized payments, please refer to [Tokenized Payments](#).



Example

Possible scenarios for category_preference parameter:

- If no category_preference is specified, the messaging copy for the **all-option approach** will be returned by default (default and recommended solution)
- If category_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 category_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 customer enters the purchase flow.

Release notes

²⁰²⁰
17) PAYMENT_DESCRIPTOR_SUBHEADER support will be added in a later release.

Response example:

```
Unset
{
  "paymentDescriptors": [
    {
      "payment_option_id": "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"
      }
    ]
  },
  {
    "payment_option_id": "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"
        }
      ]
    }
  },
  {
    "preselection_id": "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"
    }
  ]
}
}
}]

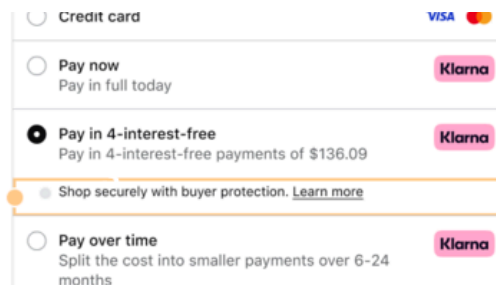
```

2.6.1.1.3.3 - 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 Klarna Payment Services, this section includes a customer protection USP and a "learn more" link.



JavaScript

```

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

```

Alternate: Static assets

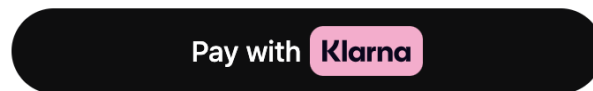
If relying on external dependencies is not feasible when building the checkout, or if you, the acquiring partner, do not own the client-side presentation of Klarna and cannot accept the `payment_option_id`, all assets (such as payment descriptors, payment subheaders, and the Klarna badge) are accessible 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.

2.6.1.1.4 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 a simple client-side integration with proactively maintained messaging. If this is not possible, an alternate flow is described in [Klarna payment button cannot be initiated](#).



We recommend merchants to change their payment button to Klarna's payment button when a customer selects Klarna, using the theme most appropriate to the Partner page.

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

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

2.6.1.1.4.2 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');
```

2.6.1.1.4.3 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	<p>This is the default value and recommended. Klarna automatically selects the best flow depending on the device:</p> <ul style="list-style-type: none"> • Mobile: REDIRECT • Desktop: Modal window if possible, fallback to REDIRECT. • Native webview (mobile app) - REDIRECT <p>Note: a <code>config.redirectUrl</code> is required in the payment request for this interaction mode</p>
ON_PAGE	<p>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.</p>
REDIRECT	<p>Only redirect flow.</p> <p>Note: a <code>config.redirectUrl</code> is required in the payment request for this interaction mode</p>

Generate a Redirect URL

The `redirectUrl` guides the customer back to your online store after a completed or canceled payment request. By embedding placeholders in the URL, Klarna can dynamically insert pertinent transaction information, ensuring the URL is comprehensive for transaction processing.

To ensure security and integrity, the server-side confirmation payment request call is required to complete a payment. 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 completes the payment, preventing hijacking.

Redirect URL example:

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

These placeholders enable the redirect URL to dynamically incorporate all necessary transaction details. Klarna replaces these placeholders with actual values prior to redirection, facilitating a smooth redirection process for partners and ensuring that all critical information is accessible for processing the transaction. Details about the available placeholders are provided 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 be initiated either directly by you, as the integrator, using the Klarna Payment Services, or implicitly through 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 unless 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 <code>redirectUrl?</code> properties can be provided.
config.payment_option_id	The identifier returned by Klarna's messaging API or Web SDK to support the preselection of payment options within the Klarna payment 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 features.



Parameter	Definition
	If Klarna's Web SDK is integrated by the partner, this identifier must be mapped to the shoppingSessionId returned by Klarna. Klarna Partners must make this value available to Partners, and allow them to pass this through to Klarna at all interactions.
merchantReference (optional)	Used for storing the customer-facing transaction number. It will be displayed to customers on the Klarna app and other communications. It will also be included in settlement reports for the purpose of reconciliation.
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 Partner. 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 customer 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

2.6.1.1.4.4 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
klarnaPaymentButton.on("click", async (paymentRequest) => {
  paymentRequest.initiate(
    {
      paymentAmount: 9999,
      currency: "EUR",
      config: {
        redirectUrl:
"https://example.com?id={klarna.payment_request.id}&token={klarna.payment_request.paymen
t_confirmation_token}"
      }
    },
    {
      interactionMode: "DEVICE_BEST"
    }
  );
});

```

Upon being redirected, the customer will enter Klarna's [payment flow](#), where they can select their payment method and provide any required information. This process may also involve signing into their Klarna account to continue with the payment.

⚠ Be mindful when sharing customer data:

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

2.6.1.1.5 Step 4: Monitor payment state and retrieve payment confirmation token

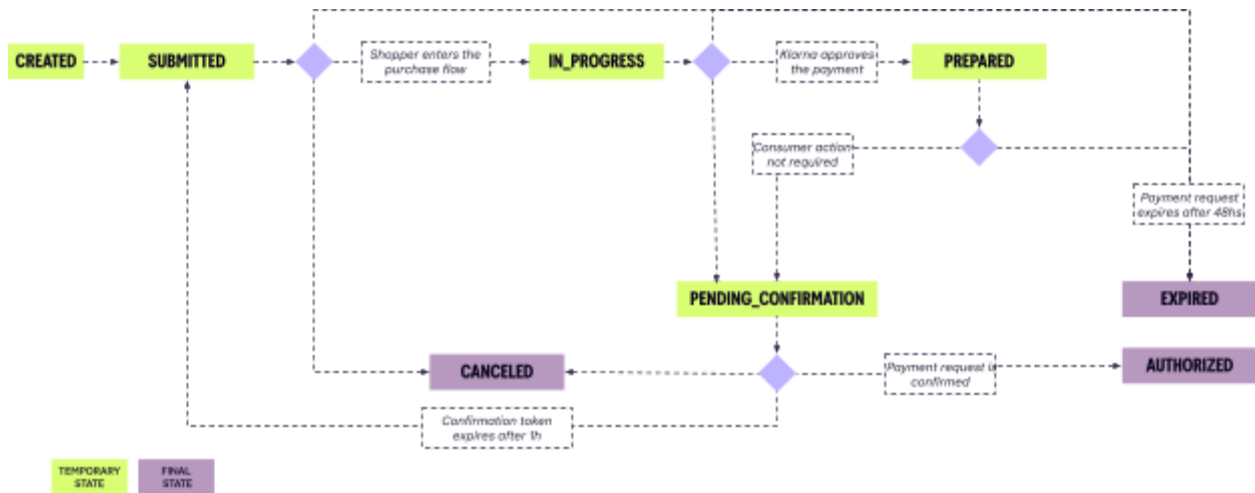
During the checkout process, the payment request will transition to various states. Find below an overview of the possible transaction states together with a transition diagram:

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 (customer is inside the purchase flow)
PREPARED	The payment request is prepared, but not yet finalized. This state can be triggered when using the <code>prepare()</code> method in the context of a multi-step checkout It must be finalized by calling <code>initiate()</code>
PENDING_CONFIRMATION	The payment flow has successfully been completed by the customer and is pending final confirmation to complete the request.
EXPIRED	The payment request has expired ($t \geq 48h$). 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. The <code>payment_confirmation_token</code> can only be CANCELED until the request is AUTHORIZED. After authorization, the cancellation is no longer possible. This is a final state.

Payment request state diagram





Once the customer successfully completes the payment in the purchase flow, the payment request will reach the `PENDING_CONFIRMATION` state, and Klarna will return a `payment_confirmation_token`. This token is a unique identifier 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 [Integration resilience](#) for more information on ensuring a robust integration.

2.6.1.1.5.1 Subscribing to webhook events

The webhook triggered when the payment request reaches the state `PENDING_CONFIRMATION` will contain the `payment_confirmation_token`. The content of the `payload{}` will differ depending on the payment request state. For information on configuring webhooks please refer to [Configure Klarna webhooks](#).

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"
    }
  }
}

```

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

2.6.1.1.5.2 Client-side payment request state updates

The `on(event, callback): void` method in the Klarna SDK is an essential tool for efficiently managing state transitions of payment requests. By registering an event handler, you can dynamically respond 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
<code>paymentRequestId</code>	String	Unique identifier of this payment request
<code>paymentRequest</code>	Object	The context that was authorized by this payment request. This is always identical to the input given by the Partner (<code>paymentRequestData</code>)
<code>state</code>	Enum	Current state of the payment request
<code>stateContext</code>	Object	State specific context for the ongoing state. The <code>paymentConfirmationToken</code> will be stored in this object once the payment request reaches the state <code>PENDING_CONFIRMATION</code>

To implement this method, follow these steps:

1. **Define your callback function** that will handle specific events related to payment requests.
2. **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)
})
```

2.6.1.1.5.3 Cancel the payment request

A payment request remains open for a period of 48 hours, which is not adjustable. Klarna recommends Partners proactively close payment requests which have not resulted in successful transactions, especially if your payment request timeout is less than 48 hours. To align 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" }
```

2.6.1.1.5.4 Handling the redirect URL

Proper handling of the redirect URL placeholders is essential when using placeholders as described in [Prepare the button click event configuration](#). Once the customer completes a transaction and is redirected back to your website, the URL will contain the actual values replacing the placeholders.

To ensure security and integrity, the server-side confirmation payment request call is required to complete a payment. 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 completes the payment, preventing hijacking.

Redirect URL example:

Request:




```
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}"
}
```

Klarna adds the content requested as below, resulting in the customer being redirected as illustrated in the response.

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

Response:


```
Unset
https://partner.example/klarna-redirect?confirmation_token=krn:payment:eu1:confirmat
ion-token:51bbf3db-940e-5cb3-9003-381f0cd731b7&request_id=bebeabea-5651-67a4-a843-10
6cc3c9616a&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, more information is available in [Confirm Payment request server side](#). Consult the [API reference](#) for a complete description of the request body parameters.

2.6.1.1.6 Step 5: Confirm Payment request server side

Once the customer successfully completes the payment request, it will transition to the PENDING_CONFIRMATION state, and Klarna will return a `payment_confirmation_token` via the authorization webhook. More information on subscribing to webhooks is available in [Subscribing to webhook events](#). The `payment_confirmation_token` is crucial for confirming the payment request and generating the `payment_transaction_id`.

For tokenized payments, the response includes a `klarna_customer` object. Within this object, you'll find the `customer_token`, which can be used for future charges on customer accounts. Further details on the tokenized payment flow are available in [Tokenized Payments](#).

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

Confirm the payment and authorize the transaction by making 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.

Consult the [API reference](#) for a complete description of the 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. Please consult [Idempotency](#) for more information on implementing idempotently.

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.

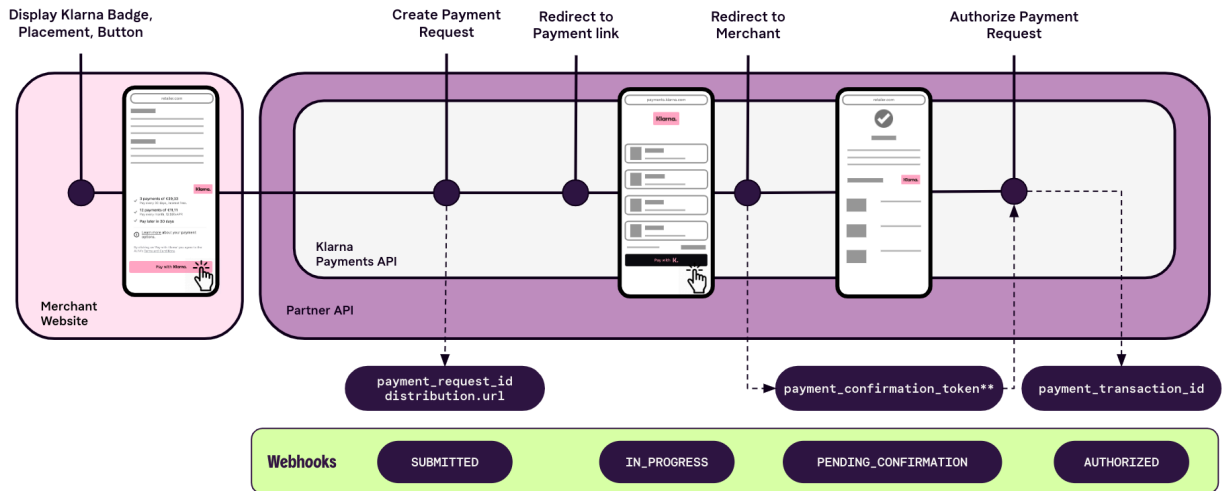
Response example:

```
Unset
{
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
    "payment_pricing": {
      "rate": {
        "fixed": 0,
        "variable": 1700000
      }
    },
    "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"
  }
}
```

2.6.1.2 Server-side initiated payment request

Overview

Use our server-side integration via REST API when your Partners are integrated with you via a server-side only integration, or you have no control over the client-side representation of Klarna in checkout. Below is an illustration of the integration flow:



* payment request expires after 48hr
** Confirmation token expires after 60m

2.6.1.2.1 Step 1: Check Klarna payment capability and display Klarna at checkout

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

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

2.6.1.2.1.1 Confirm interoperability status and select Klarna as a primary payment method

Invoke the `resolve()` 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 Klarna's availability. The method also returns the state of the shopping session registered on the customer device. Select Klarna as a primary payment method if the state is `PAYMENT_PREPARED` or `PAYMENT_PENDING_CONFIRMATION`.

JavaScript

```
// interoperabilityToken is required when Klarna.js is loaded
// on your own domain, different from the merchant domain
const interoperability = await Klarna.Interoperability.resolve({
  interoperabilityToken: "[received_interoperability_token]"
}, {
  country: "SE",
  currency: "EUR",
  paymentAmount: 10000
});
```



```

    }
  )

  switch (interoperability.status) {
    case "PAYMENT_PENDING_CONFIRMATION":
      // Confirm Klarna payment immediately, otherwise,
      // display Klarna as a selected payment method
      break;
    case "PAYMENT_PREPARED":
      // Display Klarna as a selected payment method
      break;
    case "PAYMENT_UNSUPPORTED":
      // Don't show a Klarna button or offer Klarna
      break;
    case "PAYMENT_SUPPORTED":
    default:
      // Display Klarna as a payment method
      break;
  }
}

```

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

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

1 Payment descriptor

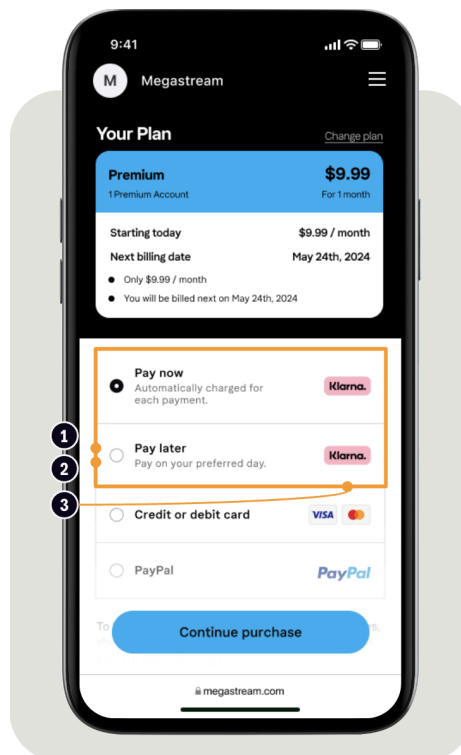
High-level call to action. Provided in Klarna response.

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.



Release notes

¹⁷ The `getPlacementContent()` method will be made available in a future release.

To achieve a flexible and global solution that allows Partners to choose their preferred presentation method, as per [How to present Klarna in checkout](#), we require you to **allow your Partners to pass the category_preference parameter at checkout**. If no or invalid preferences are provided by your Partner, the default behavior is to display all available options separately.

Additionally, allow the passing of `interoperability_token` at this point to tie the payment request to any existing client-side sessions which may have been initiated by the Partner.

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 `category_preference=KLARNA`) or
- Presented as a static payment method

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

Parameters

Parameter	Definition
<code>locale</code>	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
<code>currency (optional)</code>	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.
<code>payment_amount</code>	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.
<code>category_preference (optional)</code>	Indicates the preferred payment option. Enum: <ul style="list-style-type: none">• PAY_NOW• PAY_LATER• PAY_OVER_TIME• KLARNA
<code>message_preference (optional)</code>	Indicates the use case of checkout. Only necessary for tokenized payments. Enum: <ul style="list-style-type: none">• ECOMMERCE• SUBSCRIPTION• ON_DEMAND• IN_STORE



Parameter	Definition
subscription_interval (optional)	Indicates the cadence of the subscription if message_preference=SUBSCRIPTION, i.e. "1-MONTH"

For more detailed information on how to leverage the parameters specific to tokenized payments, please refer to [Tokenized Payments](#).

Example

Possible scenarios for category_preference parameter:

- If no category_preference is specified, the messaging copy for the **all-option approach** will be returned by default (default and recommended solution)
- If category_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 category_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 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 customer enters the purchase flow.

Release notes

NEW 17 PAYMENT_DESCRIPTOR_SUBHEADER will be added in a later release.

Response example:

```
Unset
{
```



```

"paymentDescriptors": [
  {
    "payment_option_id": "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"
        }
      ]
    }
  },
  {
    "payment_option_id": "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"
        }
      ]
    }
  },
  {
    "payment_option_id": "123xyz",
    "content": {
      "nodes": [
        {
          "name": "PAYMENT_DESCRIPTOR",

```


The `category_preference` parameter influences the type of messaging copy returned by Klarna, based on your preferences for payment options. For a detailed comparison of these approaches, please check all approaches in [Checkout structure](#).

- **Default Behavior:** If no `category_preference` is specified, the default messaging for the all-option approach will be returned. This is the recommended solution as it provides comprehensive messaging.
- **Pay Later Preference:** Specifying `category_preference=PAY_LATER` prompts Klarna to deliver messaging for the Two-option approach, highlighting Pay Later as the preferred payment option. This preference also affects other payment options.
- **Klarna Preference:** When `category_preference=KLARNA` is specified, Klarna provides the messaging for the Single-option approach.

In the response from Klarna, you will receive an array of payment descriptors. These include `PAYMENT_DESCRIPTOR`, `PAYMENT_DESCRIPTOR_SUBHEADER`, and `KLARNA_BADGE`, indexed by `payment_option_id`. This ID is crucial as it should be used when initiating the payment request. It allows you, as the acquiring partner, to preselect the corresponding payment option as the customer 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": "czxsa4324132dsaddsxzczzxs",
      "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"
  }
]
}
}
]
}

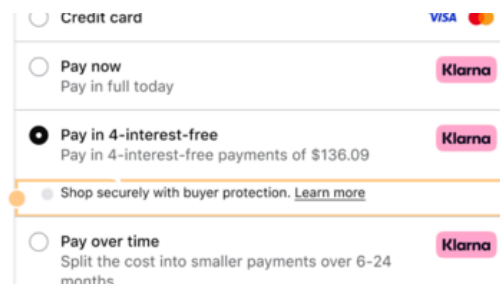
```

2.6.1.2.1.2 Display the Klarna checkout placement

Within the Klarna SDK 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. For more information on this method of displaying Klarna in checkout, refer to [Get content for Klarna's payment badge, descriptor, and subheaders](#).

Checkout Placement

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



However, for a server-side integration where the SDK is not initiated, Klarna recommends using the Partner Product Messaging API. To retrieve a placement with this API:

1. Specify a `locale` and `payment_amount` as query parameter and
2. Send a GET request to `/v1/accounts/{account_id}/payment/messaging/checkout`, 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"
  }
]
}
}

```

Alternate: 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.

2.6.1.2.2 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.	czxsa4324132dsaddsxzxzxs
interoperability.interoperability_token	Unique identifier for a shopping session.	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzaG9wcGluZ19zZXNzaW9uX2lkIjoia3JwOnNob3BwaW5nOmV1MTpzZ...
config.redirect_url	The URL provided by the integrator where the customer will be redirected after a successful purchase.	https://klarna.com?authorization_token={klarna.payment_request.payment_confirmation_token}&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,
  "interoperability": {
    "interoperability_token": "eyJ..."
  },
  "config": {
    "payment_option_id": "czxsa4324132dsaddsxzxzxs",
    "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 customer 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_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"
}

```

2.6.1.2.3 Step 3: Redirect the customer to Klarna purchase flow

After creating a payment request, the Partner should redirect the customer 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 customer will enter Klarna's [payment flow](#). This flow allows the customer to choose their payment method and input any necessary information. The process also requires the customer to log in to their Klarna account, if needed, to proceed with the payment.

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

```

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

```

2.6.1.2.4 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.

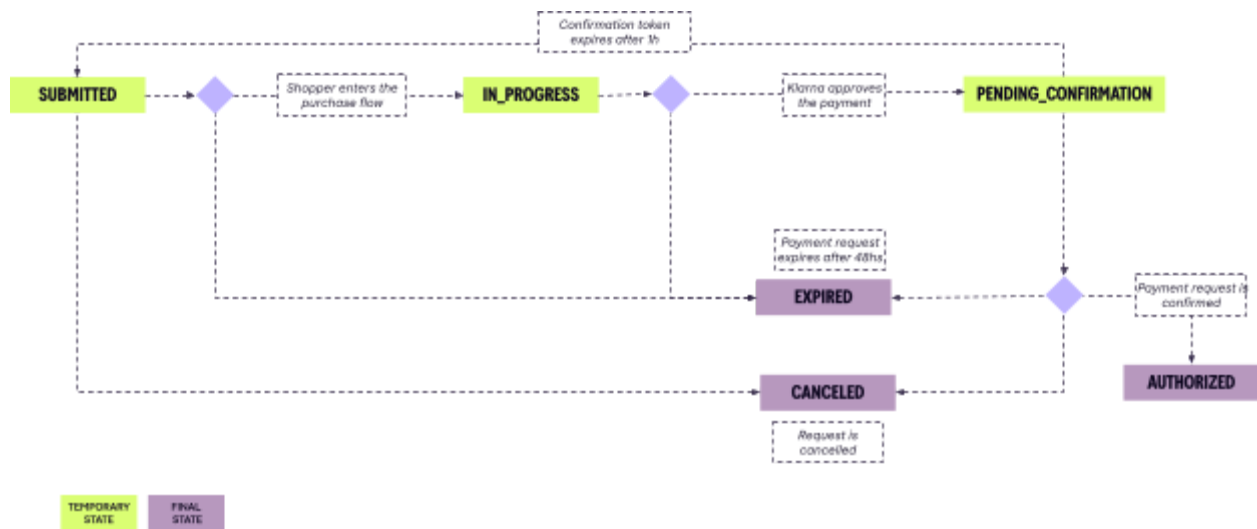
During the checkout process, the payment request will transition to various states. Find below an overview of the possible transaction states together with a transition diagram:

Payment request state definitions



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 (customer is inside the purchase flow)
PENDING_CONFIRMATION	The payment flow has successfully been completed by the customer 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. The payment_confirmation_token can only be CANCELED until the request is AUTHORIZED. After authorization, the cancellation is no longer possible. This is a final state.

Payment request state diagram



Once the customer successfully completes the payment in the purchase flow, the payment request will reach the PENDING_CONFIRMATION state, and Klarna will return a payment_confirmation_token. This token is a unique identifier 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 [Integration resilience](#) for more information on ensuring a robust integration.

2.6.1.2.4.3 Subscribing to Webhook Events

The webhook triggered when the payment request reaches the state PENDING_CONFIRMATION will contain the payment_confirmation_token. The content of the payload{} will slightly differ



depending on the payment request state. To configure webhooks for the Partner accounts please follow the guidelines in [Step 2: Configure Klarna webhooks](#).

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",
    "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"
  }
}
```

Consult the [API reference](#) for a complete description of the webhook payload.

2.6.1.2.4.2 Cancel the payment request

A payment request remains open for a period of 48 hours, which is not adjustable. Klarna recommends Partners proactively close payment requests which have not resulted in successful transactions, especially if your payment request timeout is less than 48 hours. To align 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", }
```

2.6.1.2.4.4 Placeholder in the redirect_url

The `redirect_url` directs the customer 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 a payment. 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 completes the payment, preventing hijacking.

Details on available placeholders 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
{klarna.payment_request.referrer}	URL where the authorization started.	https://partner.example/pdp

Redirect URL example:

Request:

```
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}"
}
```

Klarna adds the content requested as below, resulting in the customer being redirected as illustrated in the response.

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

Response:

```
Unset
https://partner.example/klarna-redirect?confirmation_token=krn:payment:eu1:confirmation-token:51bbf3db-940e-5cb3-9003-381f0cd731b7&request_id=bebeabea-5651-67a4-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, more information is available in [Confirm Payment request server side](#). Consult the [API reference](#) for a complete description of the request body parameters.

2.6.1.2.4.5 Read the Payment Request

The confirmation token can also be retrieved through the read payment request endpoint. This approach allows integrators to secure the necessary token to finalize the transaction. It is especially beneficial for verifying the token or acquiring it following the initial redirection flow. By leveraging both the redirect URL placeholders and the read payment request endpoint, Partners can ensure they have all the necessary tools to manage and confirm payments efficiently.

To retrieve the confirmation token or read the order in this way, send 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" }

```

2.6.1.2.5 Step 5: Confirm Payment request server side

Once the customer successfully completes the payment request, it will transition to the PENDING_CONFIRMATION state, and Klarna will return a `payment_confirmation_token` via the authorization webhook. More information on subscribing to webhooks is available in [Subscribing to webhook events](#). The `payment_confirmation_token` is crucial for confirming the payment request and generating the `payment_transaction_id`.

For tokenized payments, the response includes a `klarna_customer` object. Within this object, you'll find the `customer_token`, which can be used for future charges on customer accounts. Further details on the tokenized payment flow are available in [Tokenized Payments](#).

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

Confirm the payment and authorize the transaction by making 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.

Consult the [API reference](#) for a complete description of the 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 available in [Idempotency](#).

Request:

```

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:

```
Unset
{
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
    "payment_pricing": {
      "rate": {
        "fixed": 0,
        "variable": 1700000
      }
    }
  },

  "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"
}
```



2.6.2 Tokenized Payments


Tokenized payments with Klarna enable partners to securely store multiple payment details per customer, with the customer's consent. This feature supports subscription payments, enhances the checkout process with saved payment details in a digital wallet, and allows the combination of one-time and recurring payments.

We refer to these saved payment details as customer tokens, and the process of storing these details as tokenization. To leverage Klarna's tokenization services, Klarna Partners must adhere to data protection obligations and comply with all applicable laws and regulations as outlined in the Klarna Network Rules.

To support tokenization, an acquiring partner only needs to pass additional parameters when making a payment request. Klarna then generates a customer token that may be used for future payments. To complete future payments, a [charge token](#) request may be made with the customer token to create an order with or without customer interaction.

By default, tokens are linked to a single partner account. However, sharing tokens across multiple accounts is possible if needed.

Release notes

 Support for sharing customer tokens across multiple Partner accounts will be added in a later release.

2.6.2.1 Types of tokenized payments

Klarna's tokenized payments solution supports three main use cases:

Use case	Definition
Subscriptions	<p>A recurring transaction made at regular intervals for a product or a service.</p> <p>Subscription payments with Klarna support a broad range of subscription business models, including but not limited to free trials and non-free trials.</p>
On-demand	<p>Unscheduled transactions where a customer can store their payment details to use for payments in your website or app at a later time using their saved details.</p> <p>Klarna on-demand payments can only be used for on-demand services where the customer has a verified account with the partner and initiates the transaction themselves. . Examples of these services include ride sharing, parking apps, food delivery services and streaming media.</p>
Mixed payments	<p>Mixed payments enable customers to combine one-time purchases with additional services that require recurring payments at checkout. Customers can select any payment option for their initial purchase and tokenize another payment option for future transactions. Common applications of mixed payments include:</p> <ul style="list-style-type: none">• Purchasing a one-time product combined with a subscription or insurance program.



Use case	Definition
	<ul style="list-style-type: none"> • Transactions that incorporate a one-time product purchase along with a <i>separately charged additional service</i>. • For more information on setting up these payment options, please visit the Partner portal or contact our support team.

Consult the Klarna Network Rules for a complete description of applicable rules for tokenization.

2.6.2.2 Customer token scopes

The customer token issued by Klarna enables partners to perform actions on behalf of the customer. Each token is linked with specific scopes that define the permissible actions. If additional permissions are required beyond the initial grant, the token needs to be updated accordingly.

Scope	Definition
<code>customer:login</code>	<p>The "Customer login" scope can be used to retrieve customers' personal data from Klarna. Retrievable data includes:</p> <ul style="list-style-type: none"> • customer contact details, • transaction information • Shopping Profile data.
<code>payment:customer_present</code>	<p>The "Customer Present on Payment" scope enables transactions to be completed without further customer interaction, facilitating one-click payments and accelerated checkouts. This scope is applicable only for on-demand services where the customer has a verified account with the partner and initiates the transaction themselves.</p> <p>Klarna partners are required to implement a step-up authentication flow as documented in Customer present token charges to support interactions with the Klarna payment flow when required.</p>
<code>payment:customer_not_present</code>	<p>Tokens with the Customer Not Present on Payment scope can be used to complete transactions on behalf of the customer, initiated by the Partner.</p> <p>This scope is essential for subscription-based models where payments are pre-authorized for goods or services without needing the customer's direct approval or interaction each time.</p>

Consult the Klarna Network Rules for a complete description of applicable rules for customer token scopes.

Example

Possible scenarios for customer Token Scopes:

- If scope = CUSTOMER PRESENT is specified, Klarna will only accept **Charge Token** requests where `customer_present = true`.



- If scope = CUSTOMER NOT PRESENT is specified, Klarna will accept **Charge Token** requests for both customer_present being true or false

In case of needing to increase scopes on the customer token, the Klarna Payment flow must be leveraged to update the scopes of the issued customer token.

2.6.2.3 Creating a customer token

To create a customer token, you need to include additional parameters when you initiate the Klarna Payment flow with requests as outlined in:

- [Klarna.js integration](#)
- [Server-side only integration](#)

Upon a successful request, a customer_token is returned in the response, which can be used for future payments. Ensure you select the appropriate integration method that aligns with your partner's business model and follow the detailed steps for integration.

Presentation of Klarna as an on-demand payment method is dependent on the method chosen at checkout. For more detail on how to present Klarna in checkout, please refer to the [checkout section](#).

As in a non-tokenized payment, Klarna will return an array of payment descriptors listing the content to be displayed. These must be leveraged for presentation of Klarna in the checkout as detailed in [dynamic display approach](#) section, where the payment_option_id must be used when initiating the payment request as it allows you the corresponding payment option to be preselected within the Klarna purchase flow. Alternatively the [static display approach](#) could be used as a fallback mechanism if dynamic display cannot be accommodated.

2.6.2.3.1 Subscriptions

2.6.2.3.1.1 Step 1: Present Klarna as a subscription payment method.

When using Klarna as a payment method for subscriptions, it's crucial to include the message_preference and subscription_interval in your request as shown below:

```
JavaScript
const descriptor = klarna.Messaging.getPlacementContent({
  key: "payment-descriptors",
  payment_amount: 2000,
  locale: "fr-CA",
  message_preference: "SUBSCRIPTION",
  subscription_interval: "1-MONTH"
})
```

This setup ensures that Klarna handles the subscription details appropriately within the payment flow. Other than the inclusion of message_preference and subscription_interval, the method of



presenting Klarna during checkout remains consistent with the standard process outlined in [Online store transactions](#).

2.6.2.3.1.2 Step 2: Create payment requests for subscriptions.

Creating a payment request for subscriptions is dependent on the method chosen in [Online store transaction](#). Once the customer confirms their intention to pay with Klarna, include the following parameters when initiating the Klarna payment flow to create a customer token for a subscription:

Parameter	Definition
currency	The currency in which the transaction is made.
payment_amount	The total amount to be charged, matching the sum of the <code>subscription.subscription_plan.billing_amount</code> . This is the amount Klarna will show the customer regarding this purchase request. In a free trial, you should set the amount to NULL.
subscription.subscription_reference	Reference to this subscription, the same reference must be passed in when charging for this subscription. The payment preference is resolved using the reference. If no reference is passed, a default preference will be used when charging the subscription.
subscription.name	The name of the subscription. This will be shown to the customer in the payment flow and in the Klarna App.
subscription.subscription_plan	Specifies the billing amount, interval and interval frequency of the subscription plan. This will be shown to the customer in the payment flow and in the Klarna App. This is also used to determine eligible payment methods in the payment flow.
subscription.subscription_items	Specifies the items included in the subscription payment.
config.request_customer_token	Request a customer token to be set up, effectively returning a token on successful confirmation of the payment request. The generated token will be available under <code>state_context.customer_token</code> when the request is in state AUTHORIZED

Consult the [API reference](#) for a complete description of the request body parameters. Configurations for different subscription types are detailed below.

Release notes

 Support for subscription objects will be added in a subsequent release.



Request example for subscription costing \$9.99 USD charged monthly with a free trial

```
JavaScript
{
  "currency": "USD",
  "payment_amount": NULL,
  "subscriptions": [
    {
      "subscription_reference": "PREMIUM",
      "name": "Premium",
      "subscription_plan": {
        "billing_amount": 0,
        "interval": "MONTH",
        "interval_frequency": 1,
        "next_billing_date": "2024-02-01",
        "next_billing_amount": 999
      },
      "subscription_items": [
        {
          "name": "Premium free trial",
          "quantity": 1,
          "total_amount": 0
        }
      ]
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_customer_token": {
      "scopes": ["payment:customer_not_present"]
    },
    "redirect_url": "https://partner.example/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 customer should be redirected to.

Response example for subscription costing \$9.99 USD charged monthly with a free trial

Response 201 OK

```
JavaScript
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Content-Type: application/json

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
```



```

    "distribution": {
      "url": "https://pay.klarna.com/l/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}

```

Request example for subscription costing \$9.99 USD charged monthly without a free trial

```

JavaScript
{
  "currency": "USD",
  "payment_amount": 999,
  "subscriptions": [
    {
      "subscription_reference": "PREMIUM",
      "name": "Premium",
      "subscription_plan": {
        "billing_amount": 999,
        "interval": "MONTH",
        "interval_frequency": 1,
        "next_billing_date": "2024-02-01",
        "next_billing_amount": 999
      },
      "subscription_items": [
        {
          "name": "Premium",
          "quantity": 1,
          "total_amount": 999
        }
      ]
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_customer_token": {
      "scopes": ['payment:customer_not_present']
    },
    "redirect_url": "https://partner.example/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 customer should be redirected to.

Response example for subscription costing \$9.99 USD charged monthly without a free trial

Response **201 OK**

```
JavaScript
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Content-Type: application/json

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "distribution": {
      "url": "https://pay.klarna.com/1/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z",
}
```

2.6.2.3.1.3 Step 3: Confirm payment request server side.

After a customer completes the payment flow and the payment request transitions to the PENDING_CONFIRMATION state, Klarna will issue a payment_confirmation_token. This token is used to confirm the payment request and retrieve a customer token.

For comprehensive instructions on confirming payment requests, refer to [Payment request server side](#). Customer journeys which differ from this default flow are outlined below.

Subscriptions with free trial:

To create a subscription with a free trial, set the payment_amount:NULL as in the example below:

```
JavaScript
{
  "currency": "USD",
  "payment_amount": NULL
}
```

Upon successful confirmation, Klarna will return an HTTP 200 OK response. The response body includes the payment_transaction_id necessary for managing payment transactions and the customer_token for creating future charges. As the payment amount was NULL, it will be reflected as such as illustrated in the example response snippet below. This snippet contains only the state_context object for an on-demand confirmation response. Consult the [API reference](#) for more details.



```

JavaScript
state_context": {
  "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
  "klarna_customer": {
    "customer_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e30.Et9HFtf9R3GEMA0IIC0fFMVXY7kkTX1wr4qCyhIf58U",
    },...
  "payment_pricing": {
    "rate": {
      "fixed": 0,
      "variable": 1700000
    }
  },
  "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"
}

```

Subscriptions without free trial:

To initiate a subscription without a free trial, include a specific value in the `payment_amount` as in the request below:

```

Unset
{
  "currency": "USD",
  "payment_amount": 999
}

```

Similar to the free trial, a successful confirmation will generate an HTTP 200 OK response from Klarna. The response includes both the `payment_transaction_id` for transaction management and the `customer_token` for creating future charges. The initial charge amount will also be detailed in the response.

2.6.2.3.2 On-demand

2.6.2.3.2.1 Step 1: Present Klarna as a payment method

When leveraging Klarna as an on-demand payment method, it is crucial to include the `message_preference` parameter in the request as shown below:

```

JavaScript
const descriptor = klarna.Messaging.getPlacementContent({
  key: "payment-descriptors",

```

```

payment_amount: 0,
locale: "fr-CA",
message_preference: "ON_DEMAND"
})

```

Aside from adding `message_preference`, presenting Klarna during checkout remains consistent with the standard process described in [Online store transactions](#).

2.6.2.3.2.2 Step 2: Create payment request for on-demand

Once the customer confirms the intent to select Klarna, include the following parameters when initiating the Klarna payment flow to create a customer token for on-demand payments:

Parameter	Definition
<code>currency</code>	The currency in which the transaction is made.
<code>payment_amount</code>	<p>The total amount to be charged, matching the sum of the <code>line_items[].total_line_amount</code>.</p> <p>This is the amount Klarna will charge the customer for this purchase request. If you're only setting up a customer token you should set the amount to NULL.</p>
<code>ondemand_profile.reference</code>	<p>Reference to this on-demand profile, the same reference must be passed in when charging for this on-demand service.</p> <p>The payment preference is resolved using the reference. If no reference is passed, a default preference will be used when charging the customer token for future payments.</p>
<code>ondemand_profile.details</code>	<p>Specifies the expected average, minimum and maximum purchase amount and purchase frequency of the customer for the on-demand service.</p> <p>Used for a more precise decision on payment method eligibility within the Klarna Payment flow for future payments.</p>
<code>config.request_customer_token</code>	<p>Request a customer token to be set up, effectively returning a token on successful confirmation of the payment request.</p> <p>The generated token will be available under <code>state_context.customer_token</code> when the request is in state AUTHORIZED</p>

Consult the [API reference](#) for a complete description of the request body parameters. Configurations for different subscription types are detailed below.

Release notes



17 Support for on demand objects will be added in a later release.

Request example for on-demand payment of \$25 USD with initial purchase:

```
JavaScript
{
  "currency": "USD",
  "payment_amount": 2500,
  "ondemand_profile": {
    "ondemand_profile_reference": "mega_scooters_wallet",
    "ondemand_profile_details": {
      "average_amount": 3500,
      "minimum_amount": 100,
      "maximum_amount": 15000,
      "purchase_interval": "WEEK",
      "purchase_interval_frequency": 1
    }
  },
  "line_items": [
    {
      "name": "Scooter ride",
      "quantity": 1,
      "total_amount": 2500
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_customer_token": {
      "scopes": ["payment:customer_present"]
    },
    "redirect_url": "https://partner.example/redirect?id={klarna.payment_request.id}",
  }
}
```

Upon successful creation, Klarna responds with HTTP 201 Created. The response includes `payment_request_id` and the `state_context.distribution.url` to which the customer should be redirected.

Response example for on-demand payments with initial purchase:

Response 201 OK

```
JavaScript
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Content-Type: application/json

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
```

```

    "distribution": {
      "url": "https://pay.klarna.com/l/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z",
}

```

Request example for on-demand payment at a Partner with an AoV of \$35 USD without initial purchase:

```

JavaScript
{
  "currency": "USD",
  "payment_amount": NULL,
  "ondemand_profile": {
    "ondemand_profile_reference": "mega_scooters_wallet",
    "ondemand_profile_details": {
      "average_amount": 3500,
      "minimum_amount": 100,
      "maximum_amount": 15000,
      "purchase_interval": "WEEK",
      "purchase_interval_frequency": 1
    }
  },
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_customer_token": {
      "scopes": ["payment:customer_present"]
    },
    "redirect_url": "https://partner.example/redirect?id={klarna.payment_request.id}",
  }
}

```

Upon successful creation, Klarna responds with HTTP 201 Created. The response includes `payment_request_id` and the `state_context.distribution.url` to which the customer should be redirected.

Response example for on-demand payment at a Partner with an AoV of \$35 USD without initial purchase:

Response 201 OK

```

JavaScript
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj

```



Content-Type: application/json

```
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "distribution": {
      "url": "https://pay.klarna.com/1/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}
```

2.6.2.3.2.3 Step 3: Confirm payment request server side.

Once the customer has successfully completed the payment flow, the payment request will reach the state `PENDING_CONFIRMATION` and a `payment_confirmation_token` will be returned by Klarna that will be used to confirm the payment request and retrieve a customer token.

Consult the [Confirm Payment request server side](#) section for a complete description of how to confirm payment requests. Configurations specific to on-demand requests are detailed below.

On-demand payments with initial purchase:

```
Unset
{
  "currency": "USD",
  "payment_amount": 2500
}
```

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. More importantly, the response body will contain the `customer_token` which you will need to charge customers for future payments.

Response example for on-demand payment of \$25 USD with initial purchase:

Relevant components of the `state_context` object for a on-demand confirmation response. Consult the [API reference](#) for more details.

```
JavaScript
state_context": {
  "payment_transaction_id":
  "krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
  "klarna_customer": {
    "customer_token":
```

```

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e30.Et9Hftf9R3GEMA0IIC0fFMVXY7kkTX1wr4qCyhIf58U",
  },
  "payment_pricing": {
    "rate": {
      "fixed": 0,
      "variable": 1700000
    }
  }, ...
  "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"
}

```

On-demand payments without initial purchase:

Creation of an on-demand payment without an initial purchase is indicated by passing the "payment_amount": NULL as in the request below:

```

JavaScript
{
  "currency": "USD",
  "payment_amount": NULL
}

```

In the same manner as with an initial purchase, a successfully confirmed payment request will return both the payment_transaction_id and customer_token. As the payment_amount for a request without an initial purchase will be NULL, the response will return the same value. The customer_token returned can subsequently be used to charge for future payments.

2.6.2.3.3 Mixed payments

2.6.2.3.3.1 Step 1: Present Klarna as a payment method

To use Klarna as a payment method in mixed payments, include the message_preference parameter in your request as shown below:

```

JavaScript
const descriptor = klarna.Messaging.getPlacementContent({
  key: "payment-descriptors",
  payment_amount: 0,
  locale: "fr-CA",
  message_preference: "ECOMMERCE"
})

```


Other than the addition of `message_preference`, the presentation of Klarna in checkout does not differ from the flow outlined in the [checkout section](#).

2.6.2.3.3.2 Step 2: Create payment request for mixed payments

When the customer confirms their intention to pay with Klarna, initiate the payment request with the amount, currency of the payment and request a customer token. To offer the mixed payments flow, a few other parameters need to be included:

- Share `line_items` for the initial purchase of a one-time product or service
- Share parameters for the items that the `customer_token` will be used for with either:
 - `subscription_items` if the customer has added a subscription or for instance an insurance program with recurring charges to the basket
 - `line_items` with `supplementary_details` if the customer has added an item that requires an additional charge, for instance a one-off insurance or storage of payment details to cover for incidentals when renting a room or a car.
- The [scope](#) to request for the `customer_token` will depend on the use case that the `customer_token` is requested for.

The combination of line items as described above will result in the customer having the ability to choose any available payment option for the initial purchase, whilst a customer token will be created for the “Debit” equivalent of the payment option for future payments.


Example

The customer is buying a new console from “Xtreme games”, and adds a subscription to their cart when checking out. The selection of payment option for the new console will dictate the payment option used for future payments using the customer token:

- If the customer selects “Pay later” for the new console with Mastercard **** 1234 as their payment preference, the customer token issued will be connected to “Debit” and Mastercard **** 1234.
- Similarly, if the customer selects “Pay over time” for the new console with their bank account ING **** 123 as their payment preference, the customer token issued will be connected to “Debit” and ING *** 123.

The customer can update their payment preference at any time in the Klarna App.

Consult the [API reference](#) for a complete description of the request body parameters. Configurations for different mixed payments types are detailed below.

 The amount shared in the payment request will be charged to the customer using the initially selected payment option. If this is not desired for instance due to customer experience or legal constraints to only pay for insurances with debit, we recommend deducting the amount of the secondary item and using the `customer_token` to charge for it instead.

Release notes

17 Support for subscription_items and supplementary_details will be added in a subsequent release.

Request example for mixed payments with a subscription w. free trial:

This example reflects a Partner "Xtreme games" selling a new console, with a subscription to their online gaming service - providing the customer with a 1 month free trial.

JavaScript

```
{
  "currency": "USD",
  "payment_amount": 40000,
  "subscriptions": [
    {
      "subscription_reference": "GAME_PASS",
      "name": "Xtreme Game Pass",
      "subscription_plan": {
        "billing_amount": 0,
        "interval": "MONTH",
        "interval_frequency": 1,
        "next_billing_date": "2024-05-01",
        "next_billing_amount": 1000
      },
      "subscription_items": [
        {
          "name": "Xtreme Game Pass Free Trial",
          "quantity": 1,
          "total_amount": 0
        }
      ]
    }
  ],
  "line_items": [
    {
      "name": "Xtreme 512GB",
      "quantity": 1,
      "total_amount": 40000
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_user_token": {
      "scopes": ["payment:customer_not_present"]
    },
    "redirect_url": "https://partner.example/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 customer should be redirected to.

Response example for mixed payments with a subscription w. free trial:

Response 201 OK

JavaScript

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrNldnbEVj
Content-Type: application/json

```
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "distribution": {
      "url": "https://pay.klarna.com/l/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}
```

Request example for mixed payments with a subscription w/o. free trial:

This example reflects a Partner "Xtreme games" selling a new console, with a subscription to their online gaming service. No free trial is included.

JavaScript

```
{
  "currency": "USD",
  "payment_amount": 41000,
  "subscriptions": [
    {
      "subscription_reference": "GAME_PASS",
      "name": "Xtreme Game Pass",
      "subscription_plan": {
        "billing_amount": 1000,
        "interval": "MONTH",
        "interval_frequency": 1,
        "next_billing_date": "2024-05-01",
        "next_billing_amount": 1000
      },
      "subscription_items": [
        {
          "name": "Xtreme Game Pass",
          "quantity": 1,
          "total_amount": 1000
        }
      ]
    }
  ]
}
```

```

    }
  ]
}
],
"line_items": [
  {
    "name": "Xtreme 512GB",
    "quantity": 1,
    "total_amount": 40000
  }
],
"payment_reference": "partner-payref-1234",
"merchant_reference": "order-5678",
"config": {
  "request_user_token": {
    "scopes": ["payment:customer_not_present"]
  },
  "redirect_url": "https://partner.example/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 customer should be redirected to.

Response example for mixed payments with a subscription without a free trial:

Response 201 OK

```

JavaScript
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Content-Type: application/json

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "SUBMITTED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "distribution": {
      "url": "https://pay.klarna.com/1/ZjW2Xipgh0tjbrBGD7hzJM",
      "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
    }
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}

```

Request example for mixed payments with "on-demand":



This example reflects a Partner "Tailor Quick" selling tickets to their concert, with cancellation insurance provided by "insurance-4-U" included in the purchase (which must be paid up front, as insurance is not eligible for some payment methods).

```
JavaScript
{
  "currency": "USD",
  "payment_amount": 40000,
  "line_items": [
    {
      "name": "Tailor Quick Worldtour",
      "quantity": 1,
      "total_amount": 40000
    },
    {
      "name": "Insurance - Charged separately",
      "quantity": 1,
      "total_amount": 0,
      "supplementary_details": {
        "insurance_details": {
          "insurance_company": "Insurance-4-U",
          "insurance_type": "CANCELLATION",
          "insurance_price": 1000
        }
      }
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678",
  "config": {
    "request_user_token": {
      "scopes": ["payment:customer_not_present"]
    },
    "redirect_url": "https://partner.example/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 customer should be redirected to.

Response example for mixed payments with "on-demand":

Response 201 OK

```
JavaScript
Authorization: Basic Klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Content-Type: application/json

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
```

```

"state": "SUBMITTED",
"state_expires_at": "2024-01-01T15:00:00Z",
"state_context": {
  "distribution": {
    "url": "https://pay.klarna.com/l/ZjW2Xipgh0tjbrBGD7hzJM",
    "qr": "https://pay.klarna.com/q/ZjW2Xipgh0tjbrBGD7hzJM.svg"
  }
},
"expires_at": "2024-01-02T13:00:00Z",
"created_at": "2024-01-01T12:00:00Z",
"updated_at": "2024-01-01T13:00:00Z"
}

```

2.6.2.3.3.3 Step 3: Confirm payment request server side.

Once the customer has successfully completed the payment flow, the payment request will reach the state PENDING_CONFIRMATION and a payment_confirmation_token will be returned by Klarna that will be used to confirm the payment request and retrieve a customer token.

Consult [Confirm Payment request server side](#) for a complete description of how to confirm payment requests. Configurations for the subscription requests are detailed below.

Mixed payments with subscriptions with free trial:

Creation of a mixed payment with a subscription with a free trial is done as the request below:

```

JavaScript
{
  "currency": "USD",
  "payment_amount": 40000
}

```

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. More importantly, the response body will contain the customer_token which you will need to charge customers for future payments.

Mixed payments with subscriptions without a free trial:

Creation of a mixed payment with a subscription without a free trial is done as the request below:

```

JavaScript
{
  "currency": "USD",

```

```
  "payment_amount": 40000
}
```

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. More importantly, the response body will contain the `customer_token` which you will need to charge customers for future payments.

Mixed payments with on-demand:


Creation of a mixed payment with on-demand is done as the request below:

```
JavaScript
{
  "currency": "USD",
  "payment_amount": 40000
}
```

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. More importantly, the response body will contain the `customer_token` which you will need to charge customers for future payments.

2.6.2.4 Charging a customer token


Use the `customer_token` to create a payment request using the Partner Product API. The approach to create a payment request using the customer token depends on the use case. Select the option that fits your Partners business model and follow the integration steps.

 Please note that the type of token charge initiated also requires the adequate scope set on the customer token. Token charges with mismatching scopes may be rejected. See [Customer token scopes](#) for more details.

2.6.2.4.1 Customer present token charges

2.6.2.4.1.1 Step 1: Display Klarna as a saved payment method

Release notes

 Support for displaying Klarna as a saved payment method will be added in a subsequent release. In the meantime, we recommend showing the Klarna logo and "Klarna" as the name of the payment method.

2.6.2.4.1.2 Step 2: Charge token

When the customer is present, use the customer token to create a token charge request from your server including the following parameters in the request:

Parameter	Definition
currency	The currency in which the transaction is made.
payment_amount	The total amount to be charged.
ondemand_profile.reference	The same reference specified when creating the customer token must be passed in when charging the token to ensure the customers payment preferences will be applied correctly.
config.customer_present	Set customer_present to true to indicate that the customer is present in your checkout flow and can if need be fulfill a step-up authentication request made by Klarna.
config.redirect_url	Share a redirect_url that will be used to redirect the customer to the Klarna Payment flow if Klarna requests step-up authentication.

Consult the [API reference](#) for a complete description of the request body parameters. Configurations for a customer present token charge is detailed below.

Release notes

 Support for on demand objects will be added in a subsequent release.

Request example for Charge Token:

This example reflects a customer being charged for a Banana Split totalling \$25 USD. This purchase is initiated directly by the customer, reflected by customer_present=true.

```
Java
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Klarna-Partner-Account: K12345
Content-Type: application/json

{
  "customer_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e30.Et9HFtf9..."
  "currency": "USD",
  "payment_amount": 2500,
  "ondemand_profile": {
    "ondemand_profile_reference": "mega_scooters_wallet"
  },
  "line_items": [
    {
      "name": "Banana Split",
      "quantity": 1,
    }
  ]
}
```




```

      "total_amount": 2500
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678"
  "config": {
    "customer_present": true,
    "redirect_url": "https://partner.example/redirect?id={klarna.payment_request.id}"
  }
}

```

Response example for Charge Token:

Response: **201OK**

JavaScript

Content-Type: application/json

Klarna-Correlation-Id: a1fae73c-7a2e-4460-9b1f-b5c44d985cb7

```

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "AUTHORIZED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0"
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}

```

2.6.2.4.1.3 Step 3: Handle step-up authentication flow

In certain situations, customer input is needed when charging a customer token. This might happen when additional authentication is requested by Klarna or an issuing bank or when Klarna does not have valid payment method details on file for the customer.

In this scenario, the response from the token charge request will return a 200 status code and the status of the payment request being **SUBMITTED**. Klarna will also share a **distribution URL** to be used to redirect the user to the Klarna Payment flow.

After redirecting the customer and the customer has completed the Klarna Payment flow, a `payment_confirmation_token` is returned and the payment request must be confirmed similarly to when creating a customer token the first time, transitioning the status of the payment request to **CONFIRMED**. The `customer_token` will not generally be updated in this case.

If the payment request has not been confirmed within 24 hours, the payment request will transition to **DECLINED**. In this case, the Partner must notify their customer to return to their checkout to



complete the payment (for example, by sending an email or push notification). See more information about how to handle rejections [here](#).

2.6.2.4.2 Customer not present


2.6.2.4.2.1 Charge token

When the customer is not present, use the customer token to create a token charge request from your server including the following parameters in the request:

Parameter	Definition
currency	The currency in which the transaction is made.
payment_amount	The total amount to be charged.
subscription.subscription_reference	The same reference specified when creating the customer token must be passed in when charging the token to ensure the customers payment preferences will be applied correctly.
subscription.name	The name of the subscription. This will be shown to the customer in the Klarna App.
subscription.subscription_plan	Specifies the billing amount, interval and interval frequency of the subscription plan. This will be shown to the customer in the Klarna App. This is also used to determine eligible payment methods in the token charge.
config.customer_present	Set customer_present to false to indicate that the customer is not present in your checkout flow during the attempt and can't fulfill a step-up authentication request made by Klarna.

Consult the [API reference](#) for a complete description of the request body parameters. Configurations for a customer not present token charge is detailed below.

Release notes

 Support for subscription objects will be added in a subsequent release.

Request example:

```
Java
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrNldnbEVj
Klarna-Partner-Account: K12345
Content-Type: application/json

{
  "customer_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiYm9keS11IiwiaWF0IjoiMTY1MjM0NTY3In0",
  "currency": "USD",
```



```

"payment_amount": 999,
"subscriptions": [
  {
    "subscription_reference": "PREMIUM",
    "name": "Premium",
    "subscription_plan": {
      "billing_amount": 999,
      "interval": "MONTH",
      "interval_frequency": 1,
      "next_billing_date": "2024-08-01",
      "next_billing_amount": 999
    },
    "subscription_items": [
      {
        "name": "Premium",
        "quantity": 1,
        "total_amount": 999
      }
    ]
  }
],
"payment_reference": "partner-payref-1234",
"merchant_reference": "order-5678"
"config": {
  "customer_present": false,
  "redirect_url": NULL
}
}

```

Response example:

Response: **201OK**

JavaScript

Content-Type: application/json

Klarna-Correlation-Id: a1fae73c-7a2e-4460-9b1f-b5c44d985cb7

```

{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "AUTHORIZED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0"
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}

```

Consult [Handling rejections](#) to set up support to handle non-successful payments.



2.6.2.4.3 Support customer token rotation

Customer tokens issued by Klarna do not expire. However, to ensure backwards compatibility and enhanced security, Klarna can rotate and issue a new customer token in the response of a token charge request.

To ensure seamless continuity of using customer tokens, acquiring partners must support exchanging the stored `customer_token` whenever the token shared in the response has been updated, regardless of if it is a customer present or not present token charge.

Request example:

```
Java
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Klarna-Partner-Account: K12345
Content-Type: application/json

{
  "customer_token": "ABCDiJkci0iJIUzI1NiIsInR5cCI6IkpXVCJ9.e30.Et9HFtf9..."
  "currency": "USD",
  "payment_amount": 999,
  "subscriptions": [
    {
      "subscription_reference": "PREMIUM",
      "name": "Premium",
      "subscription_plan": {
        "billing_amount": 999,
        "interval": "MONTH",
        "interval_frequency": 1,
        "next_billing_date": "2024-08-01",
        "next_billing_amount": 999
      },
      "subscription_items": [
        {
          "name": "Premium",
          "quantity": 1,
          "total_amount": 999
        }
      ]
    }
  ],
  "payment_reference": "partner-payref-1234",
  "merchant_reference": "order-5678"
  "config": {
    "customer_present": false,
    "redirect_url": NULL
  }
}
```

When the payment request has been successfully submitted, Klarna will return an **HTTP 201**. The response body will contain the `payment_request_id` as well as the updated `customer_token` that has to be stored to be able to initiate future payments for the customer using the token.

Response example:

Response: **201 OK**



JavaScript

Content-Type: application/json

Klarna-Correlation-Id: a1fae73c-7a2e-4460-9b1f-b5c44d985cb7

```
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "AUTHORIZED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0"
  },
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}
```

2.6.2.4.4 Handling token charge rejections

A decline code is typically a HTTP error code that indicates a reason for the decline. While the code can originate from a number of sources, it often comes due to credit assessment, the issuing bank or the payment processor.

Klarna uses its own decline codes, which simplifies error handling in comparison to traditional decline codes, but defines the resolution more clearly. Use the following table to help resolve issues relating to token charge decline codes:

HTTP Status Code	State	State Reason	Definition
200	DECLINED	CUSTOMER_ACTION_REQUIRED	Temporary rejection most likely caused by insufficient funds or Klarna not having valid payment method details on file in a customer not present token charge. The rejection has been communicated to the customer by Klarna, but the Partner can instruct the customer to open the Klarna App for instructions on how to resolve the issue. The Partner can reattempt to charge the customer token again within 24 hours.
200	DECLINED	PERMANENT	Permanent rejection. You must sign up the customer again.
200	SUBMITTED		Temporary rejection most likely caused by additional authentication required by Klarna or the issuer or Klarna not having valid payment method details on file in a customer present token charge. A distribution URL has been shared to redirect the user to the Klarna Payment flow to complete the payment.



Other decline reasons are technical in nature and the Partner can retry again once confirming all details shared were correct when initiating the token charge. Consult the [API reference](#) for a complete description of the possible decline codes and resolution.

2.6.2.4.5 Webhooks for token charges

Release notes

^{4.2017} Support for webhooks for token charges will be added in a subsequent release.

2.6.2.5 Customer Token management

Using the Identity API, you have full control over customer tokens and their associated data. Each data object within a customer token is accessible via standard REST endpoints, allowing you to read the status, resources and revoke the customer token as needed.

Release notes

^{4.2017} Support for management of customer tokens will be added in Release 4.

2.6.2.5.1 Read the status of the Customer Token

Use the Management API to read the status of the customer token. The customer token has a simplified life cycle and can be updated by the **Partner** and in certain scenarios by **Klarna** or by the **Customer**, where the status of the token can be:

Parameter	Definition
ACTIVE	Represents a customer token that is fully operational and can be used to process transactions with Klarna.
REVOKED	Represents a customer token that is no longer operational in any respect. The Partner must create a new customer token to process transactions with Klarna.

Request example:

```
JavaScript
POST v1/accounts/:accountId/identity/customer-token/introspect
POST v1/identity/customer-token/introspect

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj

{
  "customer_token": ""
}
```

Response example:

Response **201OK**



```
JavaScript
{
  "status": "ACTIVE",
  "scopes": ["payment:customer_present"]
}
```

2.6.2.5.2 Read Customer Profile

In addition to reading the status of the customer token, you can also read the customer profile and payment preferences associated with the customer token.

Request example:

```
JavaScript
POST v1/accounts/:accountId/identity/customer-profile/read

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrNldnbEVj

{
  "customer_token": ""
}
```

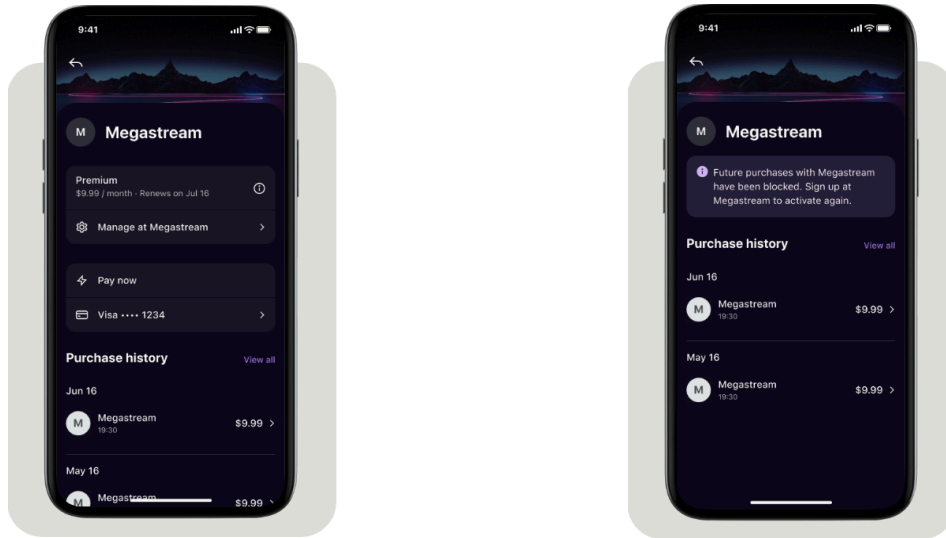
Response example:

Response 200 OK

```
JavaScript
{
  "customer_id": "krn:user:...",
  "given_name": "John",
  "family_name": "Doe",
  "address": {...},
  "payment_option": {
    "payment_option_label": "Pay in 4",
    "funding_source_label": "Visa **** 4445"
  },
  "subscriptions": [{
    "subscription_reference": "subref_12234",
    "payment_option": {
      "payment_option_label": "Pay Now",
      "funding_source_label": "Mastercard **** 7748"
    }
  }],
  "ondemand_profile": [{
    "ondemand_profile_reference": "onref_12234",
    "payment_option": {
      "payment_option_label": "Pay Now",
      "funding_source_label": "Visa **** 4445"
    }
  }]
}
```

2.6.2.5.3 Cancel Customer Token

Should a customer cancel their subscription or remove Klarna as their payment method from the Partner's digital wallet, the Partner must revoke the customer Token. Revoking the customer token ensures the correct status is displayed in all Klarna touchpoints to the customer.



Active customer token display in the Klarna App.

Revoked customer token display in the Klarna App.

Request example:

```
JavaScript
POST v1/accounts/:accountId/identity/customer-token/revoke
POST v1/identity/customer-token/revoke

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
{
  "customer_token": ""
}
```

Response example:

Response **204 OK**

2.6.2.5.4 Customer Token lifecycle webhook

Release notes

17 Support for webhooks for customer token lifecycle events will be added in Release 4.

2.6.3 More payment use cases


Klarna offers comprehensive payment solutions tailored for all types of businesses, including those with [subscription](#)-based models or [on-demand](#) services. Our omnichannel approach ensures a seamless customer experience across all platforms—online, in physical stores, and through 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 a transaction.
- [Mobile app](#): enable Klarna Payment Services in mobile apps.
- [Physical store](#): enable Klarna Payment Services in your Partners physical stores.
- [Payments on restricted businesses](#): enable Klarna for specific business models such as travel or digital services such as gaming or others.

2.6.3.1 Updating a transaction

2.6.3.1.1 Server-side updates

To update a payment request on the server-side, issue a PATCH request to [/v1/payment/{account_id}/requests/{payment_request_id}](#).

 This can only be performed when the payment request is in the **SUBMITTED** state. This restriction ensures that the customer 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_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"  
}
```

2.6.3.1.2 Client-side updates

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

⚠️ This can only be performed when the payment request is in the **SUBMITTED** state. This restriction ensures that the customer is aware of any modifications to the payment request.

Payment request

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

```
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  
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
}

```


2.6.3.3 Payments via mobile apps

Release notes

 Information on implementing Klarna in mobile apps will follow in subsequent releases.

2.6.3.4 Payments in physical store

Release notes

 Information on implementing Klarna in physical stores will follow in subsequent releases.

2.6.3.5 Payments on restricted businesses

Restricted businesses associated with Merchant Category Codes (MCCs), as outlined in the Klarna Network Rules, must provide additional data points to ensure transactions are processed in compliance with regulatory and legal standards.

As an acquiring partner, it is your responsibility to enable partners to submit supplementary data to Klarna if their MCC is categorized in Klarna Network Rules as within Group 4 of Restricted Businesses. If this information is received directly from the partner, you must ensure it is forwarded to Klarna accordingly. This process is crucial for maintaining compliance and facilitating proper transaction processing for restricted business categories.

2.6.3.5.1 Supplementary Purchase Data Requirements for Restricted businesses

Details of Supplementary Purchase Data required from Restricted MCCs:

Goods and Services	MCCs	Supplementary Purchase Data Requirements
Airlines	3000-3299	travel_reservations
Car rentals	3300-3499	travel_reservations
Hotels / Lodging	3500-3999	lodging_reservations
Cruise Lines	4411	travel_reservations
Transportation Services Not Elsewhere Classified	4789	travel_reservations
Marketplaces	5262	marketplace_seller_details
Lodging-Hotels, Motels, Resorts-not elsewhere classified	7011	lodging_reservations
Timeshares	7012	lodging_reservations
Automobile Rental Agency-Not Elsewhere Classified	7512	travel_reservations
Truck rental	7513	travel_reservations
Theatrical Producers (Except Motion Pictures), Ticket Agencies	7922	event_reservations
Bands, Orchestras and Miscellaneous Entertainers-Not Elsewhere Classified	7929	event_reservations
Amusement Parks, Carnivals, Circuses, Carnivals, Fortune Tellers	7996	event_reservations
Aquariums, Dolphinariums, Zoos and Seaquariums	7998	event_reservations

2.6.2.5.2 Send Supplementary purchase data to Klarna

The additional data required could be provided via Payments API or via Shopping Session API, see details on specific requests and properties that enables you to send supplementary data to Klarna.



2.6.3.5.2.1 Payment API

Request	Object and Properties	Details
Create payment	supplementary_purchase_data	Send a serialized json as part of the payment request creation operation.
Update payment	supplementary_purchase_data	Send a serialized json as part of the payment request update operation.
Create payment	line_items.supplementary_details .< DATA OBJECT> shipping subscriptions customer ondemand_profile	Send a structured json as part of the payment request creation operation.
Update payment	line_items.supplementary_details .< DATA OBJECT> shipping subscriptions customer ondemand_profile	Send a structured json as part of the payment request update operation.

2.6.3.5.2.2 Shopping Session API

Request	Object and Properties	Details
Create shopping session	supplementary_purchase_data.content.<DATA OBJECT>.	Send a structured json as part of the payment request creation operation.
Update shopping session	supplementary_purchase_data.content.line_items.supplementary_details.< DATA OBJECT> supplementary_purchase_data.content.shipping supplementary_purchase_data.content.subscriptions supplementary_purchase_data.content.customer supplementary_purchase_data.content.ondemand_profile	Send a structured json as part of the payment request update operation.

Example of a structured json request



```

Unset
{
  ...
  "line_items": [
    {
      ...
      "supplementary_details": {
        "travel_reservation_details": {
          "travel_type": "AIR",
          "itinerary": [
            {
              "departure": {
                "departure_airport": "CMH",
                "city": "Columbus",
                "country": "US"
              },
              "arrival": {
                "arrival_airport": "JFK",
                "city": "New York",
                "country": "US"
              },
              "carrier": "Delta Airlines",
              "segment_price": 900,
              "departure_date": "2024-01-01T12:00:00Z",
              "passenger_reference": [
                "1"
              ],
              "class": "G"
            },
            {
              "departure": {
                "departure_airport": "JFK",
                "city": "New York",
                "country": "US"
              },
              "arrival": {
                "arrival_airport": "CMH",
                "city": "Columbus",
                "country": "US"
              },
              "carrier": "Delta Airlines",
              "segment_price": 1100,
              "departure_date": "2024-01-10T09:00:00Z",
              "passenger_reference": [
                "1"
              ],
              "class": "G"
            }
          ],
          "passengers": [
            {
              "passenger_reference": "1",
              "title": "Ms",
              "given_name": "John",
              "family_name": "Doe"
            }
          ]
        }
      }
    }
  ]
  ...
}

```

2.7.2.5.2.2 Example of a serialized json request

Unset

```
{ ...  
  "supplementary_purchase_data": {"content_type": "vnd.klarna.supplementary-data.v1", "content": {"customer": {"given_name": "Klara", "family_name": "Joyce", "email": "John.doe@klarna.com"}, "merchant_account_info": {"account_id": "1234567890", "account_registered_at": "2020-01-01T12:00:00Z", "number_of_paid_purchases": 10}}}}  
}
```

Release notes

^{Buy}
17 supplementary_purchase_data object will be available in future releases.

2.6.3.6 Alternate: 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.

2.6.3.6.1 Add a button to your page:

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

2.6.3.6.2 Create the payment request:

Details on how to configure this request are outlined in [Prepare the button click](#).

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

2.6.3.6.3 Initiate payment request when customer 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](#). Additionally, consult the [payment flow](#) section for an overview of the customer journey.

2.6.4 Important payment concepts

2.6.4.1 Merchant reference

The `merchant_reference` field is a required identifier used throughout the payment process to support various partner and customer-facing activities. This parameter should correspond to the Partners customer-facing order reference. Ensuring this parameter is both consistently applied and understandable to end consumers is helpful in minimizing errands and ensuring customer satisfaction. Below are some key use cases:

- **Customer Service:** It provides a clear, recognizable identifier that customers can use during disputes, inquiries, or interactions with customer support, improving the overall user experience.
 - In addition, Klarna leverages the `merchant_reference` in all touchpoints with the consumer, ensuring clear communication and understanding within the Klarna App, email communications, and push notifications regarding order status.
- **Dispute Resolution:** `merchant_reference` provides all involved parties with an aligned understanding of the order being discussed, reducing the chances for confusion or human error for both customers and customer service agents.
- **Fallback Identifier:** When other session identifiers are unavailable, the `merchant_reference` may act as a fallback to ensure continuity and traceability within the transaction.

Merchant reference plays a key role in supporting visibility and alignment between all parties, and drives increased customer satisfaction.

2.6.4.2 Address Validation

Customer billing and shipping addresses are essential in the handling and assessment of a transaction. Validate any provided data and ensure it is handled in accordance with the data requirements outlined in the API reference and the additional guidelines provided below.

Release notes

2024
17 Klarna performs ongoing assessments to ensure that address handling continues to meet security and operational needs.

2.6.4.2.1 Client-to-Client (C2C) Integrations:

Partners must provide Klarna with all gathered customer information relevant to a purchase, ensuring compliance with any applicable privacy laws. This includes both billing and shipping



addresses. Any address provided to Klarna as part of the client-side purchase flow must be included within the confirmation request to ensure the data is aligned between parties. This validation helps protect partners and customers against fraud by detecting any changes made on the client side. If the shipping address in the initial payment request differs from the one in the confirmation, a BAD REQUEST error will be returned.

2.6.4.2.2 Server-to-Server (S2S) Integrations:

The shipping address is not required at the confirmation stage for S2S integrations as Klarna assumes server-provided data comes from a trusted, authorized source. However, if a shipping address is included in the S2S confirmation request, it must match the initially provided data.

2.6.4.2.3 Customer Data Requirements

2.6.4.2.3.1 Data Quality and Format:

Providing data in a standardized format is essential for effective fraud assessment. This includes billing and shipping addresses, email, postal code, street address, city, and phone number. If the data fails Klarna's validation or the addresses differ between requests, an error message will be generated, which should be handled as outlined in [Error handling](#).

Customer information must not be sent prior to the customer indicating a clear intention to pay with Klarna, following all applicable local privacy regulations (e.g. GDPR). This typically occurs at the time of authorization.

2.6.4.2.3.2 Unicode Support:

The following Unicode blocks are supported for all markets

- [BASIC LATIN](#)
- [LATIN_1 SUPPLEMENT](#)
- [LATIN EXTENDED A](#)
- [PHONETIC EXTENSIONS](#)

The following Unicode blocks are supported only for Greek

- [BASIC LATIN](#)
- [GREEK](#)
- [GREEK EXTENDED](#)

Additionally, characters are also matched against the following Unicode categories

- [Ll](#) (Lower case characters)
- [Lu](#) (Upper case characters)
- [Nd](#) (Decimal digit number characters)

Any characters that don't belong to the categories above are considered special characters. Further field-specific requirements are available within the [API reference](#).



2.7 Interoperability

Interoperability refers to the ability of Klarna's full product suite to work seamlessly across multiple integration paths. For an introduction to the concept of interoperability, refer to [Klarna product suite interoperability](#)

Acquiring Partners are required to build Klarna Payment Services integration such that all Klarna services in the Product Suite are able to function seamlessly together for all Partners, and Klarna Payment Services are uniformly available regardless of integration path.

2.7.1 Step 1: Grant access to Klarna's Partner portal

Klarna offers several non-payment products to Partners, including Express Checkout, On-site Messaging, and Sign in with Klarna. To enable these features, it's crucial to enable Partners to access the Partner portal after successful onboarding by their Acquiring Partner. Accessing the Partner portal enables Partners to retrieve client-side credentials and manage their brand presentation via Klarna channels. Access can be granted in one of the following ways:

Deep link integration: implement a deep link to the Partner portal from your Partner-facing admin portal. Presentation of this link must be agreed and signed off by Klarna ensuring a mutually advantageous user experience within your portal.

Alternate User Access Provision

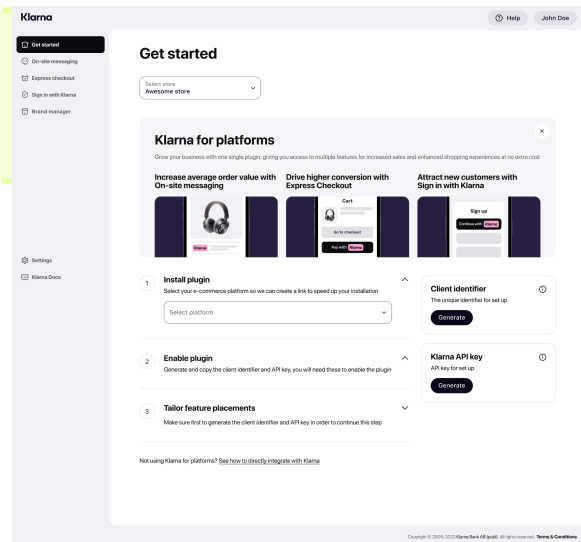
If you lack a Partner-facing admin portal that can implement a deep link, you can use the Partner portal user management API to grant Partners access to the Partner portal. Access to the portal is given by enrolling a user using their email address.

Release Notes

July 17 Portal user management via API will be introduced in future releases.

Within the Partner portal, Partners don't have the ability to manage any features which alter a transaction status, as this may be exclusively completed through their Acquiring partner. Partner functionalities available through Klarna's Partner portal include:

- **Klarna Visual Assets:** Partners can control how their brand is presented within the Klarna ecosystem by adding logos, icons, social media URLs, and display names.
- **Boost Product Access:** Enhance conversion rates by accessing Boost products such as On-site Messaging, Klarna Express Checkout, and Sign in with Klarna.



- **Partner Credential Management:** Generate client identifiers and API keys. Note that credentials generated by Partners onboarded by an Acquiring Partner are limited and do not permit access to manage payment processing.
- **Partner Campaigns:** Launch campaigns offering 0% financing or other promotions.

In some cases, additional capabilities may be available to Partners via the Partner Portal, depending on the equivalent capabilities in your own Partner-facing admin portal, which may include dispute or transaction management.

Klarna's Partner Portal access is assigned through the use of roles (outlined below), which can be developed in collaboration with Klarna.

2.7.1.1 Creating a deep link


Integrating deep linking within your Partner-facing admin portal maintains your ecosystem as the central hub through which all products are managed.

Deep links for Partner accounts should be created through a POST request to /v1/accounts/{account_id}/portal/deep-links and are valid for one-time use, expiring after 60 seconds. Upon accessing the Partner Portal using the deep link, no additional authentication is needed, and the portal session remains valid for 8 hours. You may only generate deep links for accounts which were onboarded via your services. A deep link should be requested only after the Partner expresses a desire to enter the Partner portal.

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 and debugging purposes and it should not be a personal identifier such as email. This reference is provided by you, the Acquiring partner.
roles[]	Contains the permissions to apps inside the Partner portal. Current roles: <ul style="list-style-type: none"> ● merchant:admin - provides access allowing the Partner to access all apps defined within the acquiring partner agreement. ● merchant:developer - provides access only to those apps which will assist a developer in implementing Klarna. Allows for the creation of client-side tokens and implementation of Klarna Boost features.

Release notes

 **roles** will be further defined in later releases.

2.7.1.2 Inviting a user

Directly manage users by allowing the creation of new persistent Partner Portal users with access to a previously onboarded partner account. Create a user via a POST request to /v1/accounts/{account_id}/portal/users. This triggers an email to the user containing a link to the Partner Portal account activation flow. After activation, the user can log in with the credentials set during the flow.



You can define the level of access granted to the user by setting the **roles** array:

Parameter	Definition
email	The email to be used to log into the Partner portal.
roles[]	Contains the permissions to apps inside the Partner portal. Current roles: <ul style="list-style-type: none">• merchant:admin - provides access allowing the Partner to access all apps defined within the acquiring partner agreement.• merchant:developer - provides access only to those apps which will assist a developer in implementing Klarna. Allows for the creation of client-side tokens and implementation of Klarna Boost features.

Release notes

17 **roles** will be further defined in later releases. Inviting and managing Partner portal users will also be included in later releases.

2.7.2 Step 2: Consume Klarna Interoperability Token

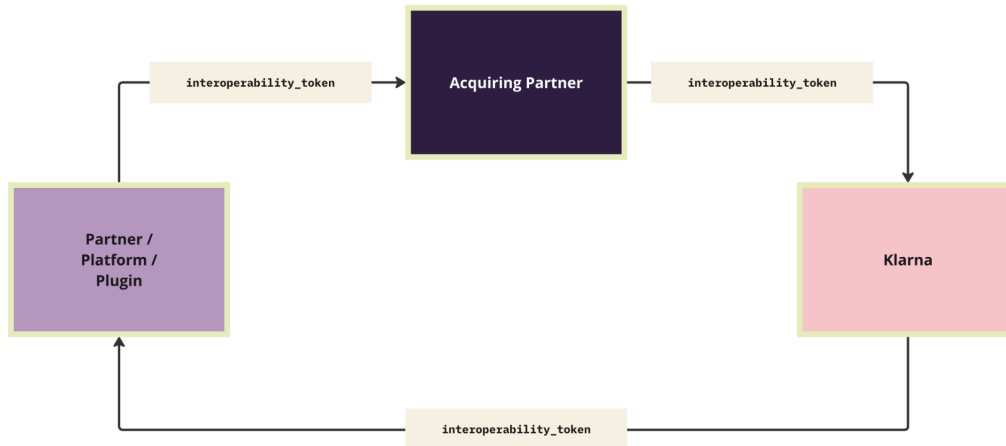
The `interoperability_token` enhances the customer experience by providing continuity between your Klarna Payment integration and the Klarna Boost experience. It allows Klarna to maintain a consistent session across different platforms to enable a framework for personalization and ensuring customers remain signed in across features, and increase your checkout conversion by sharing with you when a consumer is in a ready-to-pay state.

- **Increased flexibility** across Klarna product suite, ensuring that customers can enjoy a seamless transition between different Partner 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.

On a high level, the data flow is as follows:

1. Precondition: The Partner integrates directly with Klarna Boost Features; to hand over the consumer journey to the Acquiring Partner, they request an `interoperability_token`.
2. The Partner passes the `interoperability_token` to the Acquiring Partner.
3. The Acquiring Partner forwards the `interoperability_token` to Klarna in relevant APIs.





2.7.2.1 Consume and pass Klarna Interoperability Token

Acquiring Partners are required to accept Klarna's `interoperability_token` across all touchpoints where payment or checkout are initiated. This is achieved by allowing Partners to send the `interoperability_token` on any API endpoint that is used to trigger a payment. For example:

```

Java
POST api.acquiring-partner.com/v1/payments
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
      "interoperability_token":
      "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzaG9wcGluZ19zZXNzaW9uX2lkIjoia3JuOnNob3BwaW5nOmV1MTpzZXNzaW9uOjBlOTIyN2MwLTBjNDAtNDdkZi04YzEwLTQwMTRmYzFiZDZiZCIsInN0YXR1cyI6IiBBWU1FT1RfUFJFUeFRUQiLCJ2ZXZzaW9uIjojcjBjcmVhdGVkX2F0IjoimjAyNC0wMS0wMVQxMjowMDowMFoiLCJ1cGRhdGVkX2F0IjoimjAyNC0wMS0wMVQxMjowMDowMFoiQ.1y87kuEKnETy_UCHbTMXx1p4mViZE816okcVmfHFZoo"
    }
  }
}
  
```

⚠ The naming of the field should either be `klarna_interoperability_token` or `interoperability_token` within an explicitly Klarna specific context, so that it is easy for any Partner to identify the field and its usage. While the Acquiring Partner can introspect the token, they are required to not tamper with it and to forward it all the time.

When interacting with Klarna APIs, you must provide the `interoperability_token` whenever available. The following example demonstrates how the `interoperability_token` is provided to the Payment API:

```

Java
POST global-api.klarna.com/v1/payment/requests
  
```

```

{
  "currency": "USD",
  "paymentAmount": 17800,
  "interoperability": {
    "interoperability_token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzaG9wcGluzi19zZXNzaW9uX2lkIjoia3JuOnNob3BwaW5nOmV1MTpzZXNzaW9uOjBlOTIyN2MwLTBjNDAtNDdkZi04YzEwLTQwMTRmYzFiZDZiZCIsInN0YXR1cyI6IiBBWU1FTlRfUFJFUeFRUQiLCJ2ZXJzaW9uIjojLCJjcmVhdGVkX2F0IjojIjAyNC0wMS0wMVQxMjowMDowMFoiLCJ1cGRhdGVkX2F0IjojIjAyNC0wMS0wMVQxMjowMDowMFoiQ.1y87kuEKETy_UCHbTMx1p4mViZE816okcVmfHFZoo"
  }
}

```

2.7.2.3 Implement advanced use cases

2.7.2.3.1 Hosted checkout on your own domain instead of Partner's domain

When you use Klarna.js on your own hosted checkout domain, you're required to initialize Klarna.js with a proper interoperability context which then enables the continuity of a customer's session:

```

JavaScript
const interoperability = await Klarna.Interoperability.resolve({
  interoperabilityToken: "[received_interoperability_token]"
})

```

The integration then ensures Klarna is selected if the interoperability status is `PAYMENT_PENDING_CONFIRMATION` or `PAYMENT_PREPARED`.

```

JavaScript
switch (interoperability.status) {
  case "PAYMENT_PENDING_CONFIRMATION":
    // Confirm Klarna payment immediately when server-to-server, otherwise,
    // display Klarna as a selected payment method
    break;
  case "PAYMENT_PREPARED":
    // Confirm Klarna payment immediately when server-to-server, otherwise,
    // Display Klarna as a selected payment method
    break;
  case "PAYMENT_UNSUPPORTED":
    // Don't show a Klarna button or offer Klarna
    break;
  case "PAYMENT_SUPPORTED":
  default:
    // Display Klarna as a payment method
    break;
}

```



2.7.2.3.2 Alternate flow for Headless integrations

When the AP is processing a headless payment or has an architecture where the backend processes are driving the consumer facing user experience, it's recommended that the Interoperability Token be decoded to allow for the proper logic to ensure the optimal consumer experience.

The `interoperability_token` is a JWT token signed by Klarna. Its purpose is for the integrator to fulfill Klarna Network specific requirements without having to do a network call to Klarna's infrastructure which will add latency to a time-sensitive operation. This typically applies to a server-only integration or, in some rare cases where Klarna.js cannot be utilized. In such a scenario, the Acquiring Partner can decode the token to read its content. The decoded content of the token is a json structure as followed:

```
Java
Header:
{
  "alg": "RS256",
  "typ": "JWT",
  "cty": "application/vnd.klarna.interoperability-token-v1+json",
  "kid": "key123",
  "x5t": "abc123..."
}

Payload:
{
  "status": "PAYMENT_SUPPORTED | PAYMENT_PREPARED | PAYMENT_PENDING_CONFIRMATION",
  "shopping_session_id":
  "krn:shopping:eu1:session:0e9227c0-0c40-47df-8c10-4014fc1bd2bd",
  "payment_confirmation_token": "krn:payment:eu:confirmation-token:xxx",
  "iat": "2024-01-01T12:00:00Z"
}
```

Parameter	Definition
alg	Standard JWT header, specifies the Algorithm used to sign or verify the token.
typ	Standard JWT header, specifies the Type of the token, always set to "JWT".
cty	Standard JWT header, specifies the Content Type of the body.
kid	Standard JWT header, specifies the Key ID indicating which key was used to sign the token.

x5t	Standard JWT header, specifies a base64url-encoded SHA-1 hash of the X.509 certificate. It provides a way to identify the specific certificate associated with the public key that should be used to verify the JWT's signature.
iat	Standard JWT field which means "Issued at."
status	Specifies the interoperability status, see table below.
shopping_session_id	Specifies the shopping_session_id which must be passed on to Klarna in relevant API calls.
payment_confirmation_token (optional)	Specifies the payment_confirmation_token which must be used to confirm the Klarna Payment Request. The field is available when the status is PAYMENT_PENDING_CONFIRMATION.

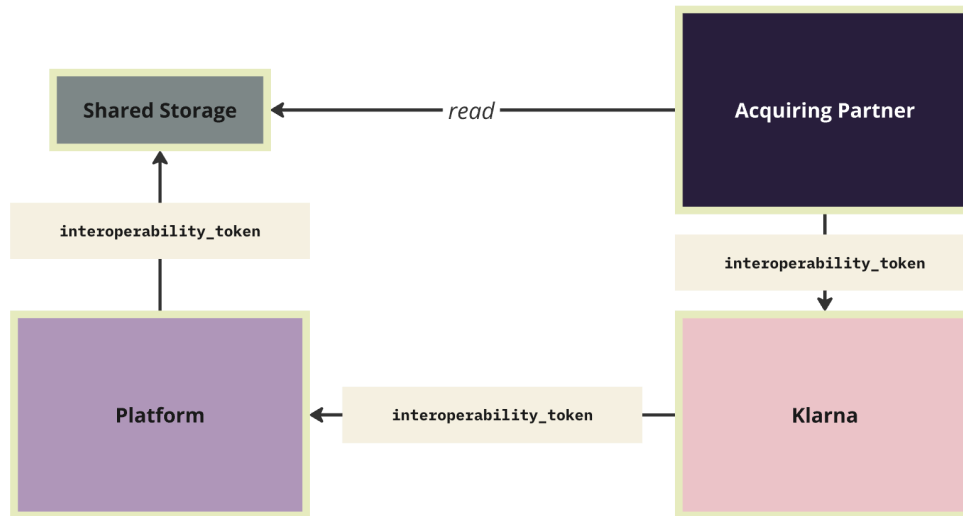
The table below different interoperability status, their use cases and expected outcomes:

Status	Use Cases	Integration Actions
PAYMENT_PENDING_CONFIRMATION	Single-step KEC	Confirm Klarna payment immediately, otherwise, display Klarna as the selected payment method.
PAYMENT_PREPARED	Multi-step KEC, SIWK	Display Klarna as the selected payment method and display "Pay with Klarna button."
PAYMENT_SUPPORTED (default)	Standard checkout	Display Klarna as a payment method
PAYMENT_UNSUPPORTED	Standard checkout	Don't show a Klarna button or offer Klarna

2.7.2.3.3 Platform plugins

If an Acquiring Partner owns Plugin solutions on platforms such as Woocommerce, Adobe Commerce, etc. they are required to retrieve the interoperability_token from the Platform's session storage. The interoperability_token is always stored with the key klarna_interoperability_token. The token should be passed to relevant Klarna API, namely Payment API and Messaging API.





Key touchpoints

- **On-site messaging:** if On-site messaging is being used, passing the `interoperability_token` will tailor the messages based on the customer's previous interactions and behaviors across the Klarna network, providing a personalized and cohesive shopping experience.
- **Payment-specific calls:** during any payment-specific calls, passing the `interoperability_token` helps link the transaction to a specific user session. This is crucial for tracking the customer's journey through different payment stages and for supporting advanced security measures like fraud detection.

2.7.3 Step 3: Consume Klarna Interoperability Data

While utilization of the `interoperability_token` is our preferred choice for interoperability, Integrating Partners don't always have the engineering capabilities to leverage the full potential of Klarna API. Additionally, an integration with Klarna API might introduce added latency to their time-sensitive operations. To balance between interoperability and complexity of the integration, an Acquiring Partner should support consuming and passing of `interoperability_data`. A prominent use case of this is submission of Supplementary Purchase Data.

Acquiring Partners should accept Klarna's `interoperability_data` across all APIs where payment or checkout is initiated. They do this by allowing a Partner to send the `interoperability_data` on any API endpoint that is used to trigger a payment. For example:

```
Java
POST api.acquiring-partner.com/v1/payments
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
```

```

        "interoperability_data":
    {"content_type":"vnd.klarna.supplementary-data.v1","content":{"customer":{"given_name":"Klara","family_name":"Joyce","email":"klara.joyce@klarna.com","merchant_account_info":{"account_id":"1234567890","account_registered_at":"2020-01-01T12:00:00Z","number_of_paid_purchases":10}}}}
    }
}
}

```

⚠ The naming of the field should either be `klarna_interoperability_data` or `interoperability_data` within an explicitly Klarna specific context, so that it is easy for any integrating partner to identify the field and its usage.

When interacting with Klarna Payment API, an Acquiring Partner must provide the `interoperability_data`:

```

Java
POST global-api.klarna.com/v1/payment/requests
{
  "currency": "USD",
  "paymentAmount": 17800,
  "interoperability": {
    "interoperability_data":
    {"content_type":"vnd.klarna.supplementary-data.v1","content":{"customer":{"given_name":"Klara","family_name":"Joyce","email":"klara.joyce@klarna.com","merchant_account_info":{"account_id":"1234567890","account_registered_at":"2020-01-01T12:00:00Z","number_of_paid_purchases":10}}}}
    }
  }
}

```

2.7.4 Step 4: Consume Klarna Payment Confirmation Token and confirm Klarna payments

Support of the `payment_confirmation_token` also allows your Partners to benefit from Klarna Express checkout. The `payment_confirmation_token` is provided by Klarna in response to a successfully completed express checkout transaction. Enabling Partners to pass this token via your services allows for seamless integration with Express checkout. The `payment_confirmation_token` is also available inside `interoperability_token` when there is a payment request associated with the user's session that is ready to be confirmed.

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

- **Endpoint requirements:** establish at least one endpoint that allows Partners 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:** use the confirmation token to confirm details of the payment via Klarna's Partner product API. Verify that the parameters of the payment fall within those supported by your integration. If the transaction is outside of supported parameters or will otherwise not be confirmed, delete the confirmation token and present the rejection to the Partner so they can inform the end customer.
- **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 Express checkout.
- **Seamless integration:** ensure there is no distinction in handling or behavior between transactions completed using 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 customer 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 Partner's integration process. This setup streamlines the transaction flow by securely encapsulating payment details.
- **Tokenization or on-demand flow:** utilizing any tokenized payment flow is an excellent opportunity to introduce the payment confirmation token. This approach is beneficial as Partners 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.

Should completion of a transaction with a provided confirmation token fail or need to be rejected, it is essential that the Partner or partner delete the Express checkout/WebSDK payment request immediately and update the customer on the status of their transaction.



2.7.5 Step 5: Consume Klarna Customer Token

Tokenized payments with Klarna enable partners to securely store multiple payment details per customer, with the customer's consent, for more information refer to [Tokenized Payments](#).

Support of the `customer_token` allows an Acquiring Partner to enable recurring payment seamlessly. An example is a customer successfully signing up using "Sign in with Klarna" and allowing the Integrating Partner to charge their account. The Integrating Partner simply registers the Klarna Customer Token with the Acquiring Partner and does not have the customer going through the payment tokenization flow.

To effectively integrate and use the `customer_token`, adhere to the following streamlined process:

- **Endpoint requirements:** establish at least one endpoint that allows Partners to register their Klarna Customer Tokens in exchange for the Acquiring Partner's token.
- **Token charge:** use the registered `customer_token` when the Integrating Partner calls to charge the connected Acquiring Partner's token.

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

Key touchpoints

- **Token charge:** allowing Integrating Partners to register and use their Klarna Customer Tokens directly will enable seamless customer experience where they don't need to go through multiple processes of tokenization with Klarna for a given Partner.
-

2.8 Managing Payment Transactions

The Payment Transaction API allows Klarna's Partners to track the lifecycle of a payment transaction and perform all post-purchase operations including transaction updates, captures, and refunds.

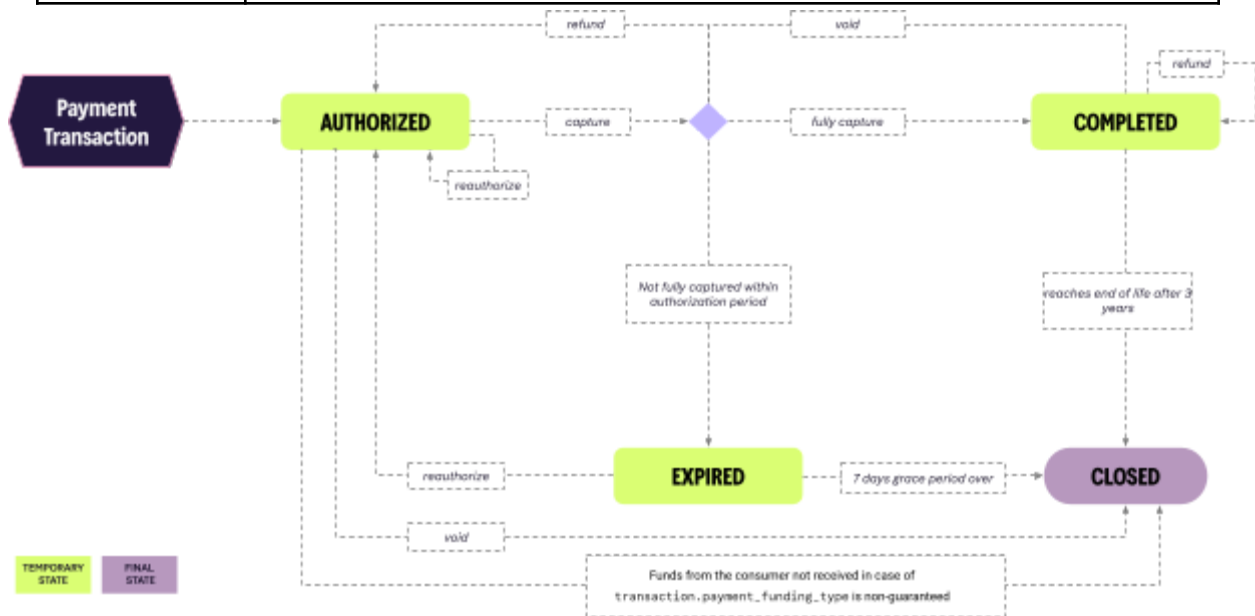
Payment transaction state definitions

The payment transaction will transition to various states during its full lifecycle, with each state representing a specific phase in the payment process dictating what actions can be taken and what limitations are in place. Below you will find an overview of the possible payment transaction states together with a state transition diagram.

State	Definition
AUTHORIZED	Represents a payment transaction that has an authorized amount remaining to be captured. It is awaiting further actions such as capture, refund, update, or void.
EXPIRED	Represents a payment transaction that has reached its lifespan without being completed. A transaction expires if it is not fully captured within a set period of time. By default, an expired payment transaction transitions to the 'CLOSED' state if it is not reauthorized within 7 days after the expiry unless otherwise agreed by Klarna.



State	Definition
COMPLETED	<p>Represents a payment transaction that has been finalized through a funds transfer. A payment transaction is considered completed when no authorized amount remains, either because it is fully captured or was partially captured and has since expired.</p> <ul style="list-style-type: none"> 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.
CLOSED	<p>Represents a payment transaction that has reached its definitive conclusion, or end of life. In this state, no further operations, including refunds, can be completed.</p> <ul style="list-style-type: none"> A payment transaction that expires before completion is deemed closed 7 days after expiry. A completed payment transaction transitions to closed after 3 years. Upon closure, no refunds can take place. A non-guaranteed payment transaction transitions to closed when not receiving the customer's funds after a specific period of time (which varies on the selected payment option).



Limitations to Payment Transaction Management

All operations that perform an action on a Payment Transaction are limited, and can only be performed 200 times each. The total number of actions performed on a given Payment Transaction may not exceed 500 operations. Read Payment Transaction is excluded from these restrictions.

Exceeding these limitations will result in a 403 response.

2.8.1 Read and update payment transactions

At any point in the payment transaction lifecycle you can use the Klarna Transactions API to retrieve detailed information about a payment transaction. You are also able to use the API to update the



transaction with additional details such as payment amount, Partner references and shipping addresses, as long as the payment transaction is not expired, completed or closed.

2.8.1.1 Read Payment Transaction

This request returns essential transaction details such as the payment amount, currency, line items, customer, shipping, pricing information and the state of the Payment Transaction. Read the payment transaction directly by sending a GET request to /v1/accounts/{account_id}/payment/transactions/{payment_transaction_id}.

Response example (State *Authorized*)

```
Unset
{
  "payment_transaction_id":
  "krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
  "currency": "USD",
  "payment_amount": 2100,
  "acquiring_channel": "ECOMMERCE",
  "merchant_reference": "merchant-order-reference-5678",
  "payment_transaction_reference": "payment-transaction-reference-1234",

  "payment_funding": {
    "payment_funding_type": "GUARANTEED",
    "payment_funding_state": "FUNDED"
  },
  "line_items": [
    {
      "name": "Black fountain pen",
      "quantity": 2,
      "total_amount": 2000,
      "product_url": "https://merchant.example/product/black-fountain-pen-1234567890",
      "image_url": "https://merchant.example/image/1234567890.png",
      "product_id": "1234567890"
    },
    {
      "name": "Sales tax",
      "quantity": 1,
      "total_amount": 100,
      "total_tax_amount": 100
    }
  ],
  "shipping": [
    {
      "recipient": {
        "given_name": "Klara",
        "family_name": "Angel"
      },
      "address": {
        "street_address": "629 N High St",
        "street_address2": "Suite 300",
        "postal_code": "43215",
        "city": "Columbus",
        "region": "OH",
        "country": "US"
      }
    }
  ]
}
```


```

    }
  },
  ],
  "captures": [],
  "refunds": [],
  "chargebacks": [],
  "state": "AUTHORIZED",
  "state_reason": "AUTHORIZED",
  "created_at": "2024-01-01T13:00:00Z",
  "expires_at": "2024-01-29T13:00:00Z",
  "original_authorization_amount": 2100,
  "remaining_authorization_amount": 2100,
  "payment_pricing": {
    "rate": {
      "fixed": 100,
      "variable": 1200000
    },
    "details": {
      "price_plan_id":
      "krn:partner:global:pricing:payments:price-plan:2a3ced5d-270b-319d-7aa4-a4bcf3a2f4b6",
      "program_id":
      "krn:partner:global:pricing:payments:program:a5cd46f7e-573d-4b02-bab8-1f1c0e343291",
      "price_plan_rate": {
        "fixed": 100,
        "variable": 1200000,
        "microtransaction_cap": 3100000
      },
      "discount": {
        "fixed": 0,
        "variable": 0
      },
      "microtransaction_cap_applied": false,
      "evaluated_at": "2024-01-01T13:00:00Z"
    }
  }
}
}
}

```

2.8.1.1.1 Non-guaranteed transactions

Release notes

 17 Support for non-guaranteed transactions will be added in a later release.

The majority of payment solutions presented by Klarna are “guaranteed”, meaning Klarna ensures payment will be made to Klarna Partner regardless of the customer's actions provided the Klarna Partner adheres to Klarna’s terms and conditions. Klarna assumes much of the customer risk inherent in the transaction.

Non-guaranteed payments are transactions where the risk if the customer fails to pay falls to the Partner. Klarna does not cover the Partner for any losses due to non-payment by the customer. These are indicated in the parameter `payment_funding_type`. For transactions where the returned

payment_funding_type is NON_GUARANTEED, the object "payment_funding" in response to the Confirm Payment Request along with the details of the Payment transaction becomes relevant.

```
Unset
"payment_funding": {
  "payment_funding_type": "GUARANTEED | NON_GUARANTEED"
  "payment_funding_state": "PENDING | FUNDED | EXPIRED"
}
```

This object helps to communicate the status of payment for those transactions where the risk profile falls outside of Klarna's guaranteed payment.

Details of the possible payment_funding_states and expected action are outlined below:

State	Definition	Expected Action
PENDING	The customer has completed the authorization flow but the funds have not yet been confirmed by Klarna.	The transaction is not funded, and Partners should be encouraged to withhold fulfillment until the state has been updated to FUNDED. Any transaction fulfilled while in this state may be subject to chargeback if Klarna is unable to retrieve funds from the customer.
FUNDED	The Klarna Partner should consider the transaction as "paid". For guaranteed payments, this means Klarna will cover the amount of the transaction toward the Klarna Partner regardless of the actions of the customer. For non-guaranteed payments, this means the customer themselves has covered the debt towards Klarna and Klarna has confirmed the funds.	The transaction should be captured on fulfillment towards the end customer.
EXPIRED	Klarna has not received the customer's funds for a non-guaranteed payment transaction after a specific period of time (which varies on the selected payment option).	Partners should stop or cancel the fulfillment of this Payment Transaction.

The Klarna Partner will be informed about the transition from "PENDING" to "FUNDED" as well as the transition from "PENDING" to "EXPIRED" with a "payment.funding.state-change" webhook.

Payload example

The following example reflects the payload structure for payment.funding.state-change event:




```

Unset
{
  "metadata": {
    ...
  },
  "payload": {
    "payment_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "payment_funding": {
      "payment_funding_type": "NON_GUARANTEED"
      "payment_funding_state": "FUNDED"
      "previous_payment_funding_state": "PENDING",
      "payment_funding_state_reason": "PAYMENT_SECURED"
    }
  }
}

```

Consult the [API reference](#) for a complete description of the response body parameters.

2.8.1.2 Update Partner references

This operation should be performed when there is a need to change either the `payment_transaction_reference` or the `merchant_reference`. The `merchant_reference` stores the customer-facing transaction identifier, displayed on the Klarna app and other customer touchpoints, and included in settlement reports for reconciliation. The `payment_transaction_reference` references the payment or equivalent resource created on your side and is exposed in Payment Transaction webhooks for correlating your resource with the Klarna Payment Transaction. These fields are essential for aligning your internal records with Klarna's transaction records and ensuring accurate tracking and reconciliation.

Request example:

```

Unset
{
  "merchant_reference": "new-merchant-order-reference-5678",
  "payment_transaction_reference": "new-payment-transaction-reference-1234"
}

```

Response example:

```

Unset
{
  "payment_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
  "currency": "USD",
  "payment_amount": 2100,
  "acquiring_channel": "ECOMMERCE",
  "merchant_reference": "new-merchant-order-reference-5678",
  "payment_transaction_reference": "new-payment-transaction-reference-1234",
  "line_items": [...],
  "shipping": [...],
  "captures": [],
}

```

```


"refunds": [],
"chargebacks": [],
"state": "AUTHORIZED",
"state_reason": "AUTHORIZED",
"created_at": "2024-01-01T13:00:00Z",
"updated_at": "2024-01-02T13:00:00Z",
"expires_at": "2024-01-29T13:00:00Z",
"original_authorization_amount": 2100,
"remaining_authorization_amount": 2100,
"customer_profile": {...},
"payment_pricing": {...}
}

```

Consult the [API reference](#) for a complete description of the response body parameters.

2.8.1.3 Reauthorize with new payment amount and line items


This operation should be performed when the `payment_amount` or `line_items` are changed.

 *Note that this action triggers a second fraud assessment, which may be rejected. This action may not be performed more than 200 times. Exceeding this will result in a 403 response.*

This operation may only be performed:

- Before the payment transaction expires.
- If the payment transaction has expired but is still within the 7-day grace period, you can extend the authorization period to reauthorize with a new payment amount.
- Before the payment transaction is fully captured.

Release notes

 **17** Re-authorize feature will be made available in Release 4.

Request example:

```

Unset
{
  "payment_amount": 4200,
  "line_items": [
    {
      "name": "Black fountain pen",
      "quantity": 2,
      "total_amount": 4000,
      "product_url": "https://merchant.example/product/black-fountain-pen-1234567890",
      "image_url": "https://merchant.example/image/1234567890.png",
      "product_id": "1234567890"
    },
    {
      "name": "Sales tax",
      "quantity": 1,

```

```

      "total_amount": 200,
      "total_tax_amount": 200
    }
  ]
}

```

Response example:

```

Unset
{
  "result": "AUTHORIZED",
  "payment_transaction": {
    "payment_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
    "currency": "USD",
    "payment_amount": 4200,
    "acquiring_channel": "ECOMMERCE",
    "merchant_reference": "merchant-order-reference-5678",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "line_items": [
      {
        "name": "Black fountain pen",
        "quantity": 2,
        "total_amount": 4000,
        "product_url": "https://merchant.example/product/black-fountain-pen-1234567890",
        "image_url": "https://merchant.example/image/1234567890.png",
        "product_id": "1234567890"
      },
      {
        "name": "Sales tax",
        "quantity": 1,
        "total_amount": 200,
        "total_tax_amount": 200
      }
    ],
    "shipping": [...],
    "captures": [],
    "refunds": [],
    "chargebacks": [],
    "state": "AUTHORIZED",
    "state_reason": "REAUTHORIZED",
    "created_at": "2024-01-01T13:00:00Z",
    "updated_at": "2024-01-02T13:00:00Z",
    "expires_at": "2024-01-29T13:00:00Z",
    "original_authorization_amount": 2100,
    "remaining_authorization_amount": 2100,
    "customer_profile": {...},
    "payment_pricing": {...}
  }
}

```

2.8.1.4 Reauthorize with new shipping details

This operation should be performed when the shipping information is changed. 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.

⚠️ Note that this action triggers a second fraud assessment, which may be rejected. This action may not be performed more than 200 times. Exceeding this will result in a 403 response.

Reauthorizing with updated shipping details is essential due to the following reasons:

- **Fraud Assessment:** Performing a second risk evaluation with the new shipping information helps to identify and mitigate potential fraudulent activities, thereby securing the transaction.
- **Disputes:** Maintaining accurate and updated shipping information supports dispute resolution by providing clear and documented proof of the shipping details, which can be referenced to customers to resolve any issues.

Release notes

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

Request example:

```
Unset
{
  "shipping": [
    {
      "recipient": {
        "given_name": "Jane",
        "family_name": "Doe"
      },
      "address": {
        "street_address": "629 N High St",
        "street_address2": "Suite 302",
        "postal_code": "43215",
        "city": "Columbus",
        "region": "OH",
        "country": "US"
      }
    }
  ]
}
```

Response example:

```
Unset
{
  "result": "AUTHORIZED",
  "payment_transaction": {
```

```

    "payment_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
    "currency": "USD",
    "payment_amount": 4200,
    "acquiring_channel": "ECOMMERCE",
    "merchant_reference": "merchant-order-reference-5678",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "line_items": [...],
    "shipping": [
    {
      "recipient": {
        "given_name": "Jane",
        "family_name": "Doe"
      },
      "address": {
        "street_address": "629 N High St",
        "street_address2": "Suite 302",
        "postal_code": "43215",
        "city": "Columbus",
        "region": "OH",
        "country": "US"
      }
    }
  ],
    "captures": [],
    "refunds": [],
    "chargebacks": [],
    "state": "AUTHORIZED",
    "state_reason": "REAUTHORIZED",
    "created_at": "2024-01-01T13:00:00Z",
    "updated_at": "2024-01-02T13:00:00Z",
    "expires_at": "2024-01-29T13:00:00Z",
    "original_authorization_amount": 2100,
    "remaining_authorization_amount": 2100,
    "customer_profile": {...},
    "payment_pricing": {...}
}

```

Consult the [API reference](#) for a complete description of the response body parameters.

2.8.1.5 Extend authorization period

This action should be taken when the existing authorization is nearing expiration and the payment cannot be completed within the current period. This approach ensures there is enough time to finalize the payment transaction without complications. Be aware that a payment transaction is considered CLOSED 7 days after its expiration and cannot be updated thereafter.

⚠️ Note that this action triggers a second fraud assessment, which may be rejected. This action may not be performed more than 200 times. Exceeding this will result in a 403 response.

When extending the authorization period, specify the number of days using the `extension_days` parameter, with a maximum extension of 180 days. It is crucial to remember that the total

authorization period for a payment transaction cannot exceed 360 days from its creation date. Any attempt to extend beyond this limit will result in rejection.

Request example:

```
Unset
{
  "extension_days": 7
}
```

Response example:

```
Unset
{
  "result": "AUTHORIZED",
  "payment_transaction": {
    "payment_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
    "currency": "USD",
    "payment_amount": 4200,
    "acquiring_channel": "ECOMMERCE",
    "merchant_reference": "merchant-order-reference-5678",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "line_items": [...],
    "shipping": [...],
    "captures": [],
    "refunds": [],
    "chargebacks": [],
    "state": "AUTHORIZED",
    "state_reason": "REAUTHORIZED",
    "created_at": "2024-01-01T13:00:00Z",
    "updated_at": "2024-01-02T13:00:00Z",
    "expires_at": "2024-02-05T13:00:00Z",
    "original_authorization_amount": 2100,
    "remaining_authorization_amount": 2100,
    "customer_profile": {...},
    "payment_pricing": {...}
  }
}
```

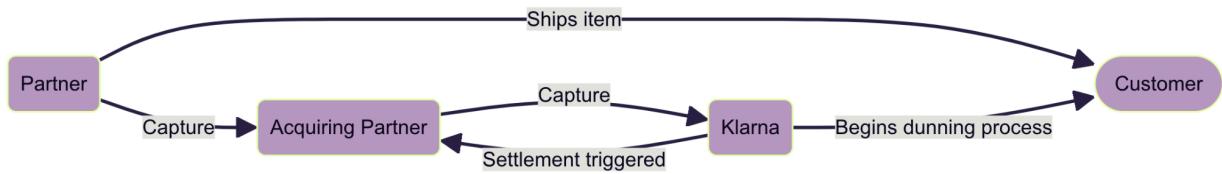
2.8.2 Capturing a Payment Transaction

This operation should be performed when the transaction, whether physical goods or intangible goods, is ready to be fulfilled. It allows for capturing either the full or partial payment amount. During capture, for physical goods you can include the shipping details along with the line items to provide a better tracking experience to customers and to improve record-keeping.

 *Klarna supports a maximum of 200 captures on any given transaction.*

The expected outcomes of a capture on all involved parties are outlined in the diagram below:





After a payment is captured and the physical goods are shipped, maintaining active communication with the customer is critical to enhance customer trust and support. Detailed shipping information is a central element of the post-purchase communication with the customer to enable:

- Elevated customer experience:
 - detailed shipping information reassures customers that their order has been processed and is on its way.
 - `tracking_number` and `tracking_url` allow customers to monitor their delivery status at any time either via Partner website, the carrier system as well as see the tracking lifecycle in the Klarna app.
- Security and accountability:
 - `shipping_type_attributes` specify the additional requirements for the delivery, for example "SIGNATURE_REQUIRED", "IDENTIFICATION_REQUIRED". This enables customers to know about for instance if they need to be present to receive the delivery.
 - Knowing the `shipping_carrier` (e.g., "DHL") helps customers identify the service responsible for their package, which can aid in troubleshooting any issues by reaching out to the corresponding support services. Additionally this information also enables Klarna to locate the tracking details.
- Improved customer support:
 - When customers have immediate access to shipping details, it reduces the likelihood of them contacting customer support for updates.
 - Real-time access allows both customers and support teams to promptly address and resolve any delivery concerns.

Ensure that the total capture amount does not exceed the payment amount unless you have reauthorized your payment transaction to a higher payment amount.

For non-guaranteed transactions, capture must not be triggered until the `funding_state` has reached "COLLECTED". If a transaction is captured prior to transitioning to that state and funds are not able to be collected by Klarna, this may result in a chargeback. More details available in [Non-guaranteed transactions](#).

2.8.2.1 Full capture

When performing a full capture, the entire value of the payment transaction should only be captured when you are ready to fulfill all goods or services included in the order.

Consult the [API reference](#) for a complete description of the request body parameters.

2.8.2.2 Partial capture

A partial capture is used when the items in the transaction are provisioned in parts. In this case, you capture only the part of the payment amount corresponding to the items being provisioned at that



time. Klarna supports a maximum of 200 partial captures on a given transaction. This process allows for multiple partial captures until the total payment amount is received.

Consult the [API reference](#) for a complete description of the request body parameters.

Request example:

```
JavaScript
{
  "capture_amount": 5000,
  "capture_reference": "capture-reference-123",
  "line_items": [
    {
      "name": "Oversized vintage trabant tee",
      "quantity": 2,
      "total_amount": 5000,
      "product_url":
"https://merchant.example/product/oversized-vintage-trabant-tee-3344556677",
      "image_url": "https://merchant.example/image/3344556677.png",
      "product_id": "1234567890"
    }
  ],
  "shipments": [
    {
      "delivery": {
        "tracking_number": "DHL123456789",
        "tracking_url":
"https://www.dhl.com/en/express/tracking.html?AWB=DHL123456789&brand=DHL",
        "shipping_carrier": "DHL",
        "shipping_type": "TO_DOOR",
        "shipping_type_attribute": "SIGNATURE_REQUIRED",
        "estimated_delivery_date": "2024-01-05"
      },
      "return": {
        "tracking_number": "DHL987654321",
        "tracking_url":
"https://www.dhl.com/en/express/tracking.html?AWB=DHL987654321&brand=DHL",
        "shipping_carrier": "DHL"
      }
    }
  ]
}
```

Response example:

```
Unset
{
  "payment_capture_id":
"krn:payment:eu1:transaction:bc42b6ff-b222-463c-b4b2-d8d6a82e0162:capture:1",
  "capture_amount": 5000,
  "capture_reference": "capture-reference-123",
  "captured_at": "2024-01-02T13:00:00Z",
  "line_items": [
```



```

    {
      "name": "Oversized vintage trabant tee",
      "quantity": 2,
      "total_amount": 5000,
      "product_url":
"https://merchant.example/product/oversized-vintage-trabant-tee-3344556677",
      "image_url": "https://merchant.example/image/3344556677.png",
      "product_id": "1234567890"
    }
  ],
  "shipments": [
    {
      "delivery": {
        "tracking_number": "DHL123456789",
        "tracking_url":
"https://www.dhl.com/en/express/tracking.html?AWB=DHL123456789&brand=DHL",
        "shipping_carrier": "DHL",
        "shipping_type": "TO_DOOR",
        "shipping_type_attribute": "SIGNATURE_REQUIRED",
        "estimated_delivery_date": "2024-01-05"
      },
      "return": {
        "tracking_number": "DHL987654321",
        "tracking_url":
"https://www.dhl.com/en/express/tracking.html?AWB=DHL987654321&brand=DHL",
        "shipping_carrier": "DHL"
      }
    }
  ]
}

```

2.8.2.3 Autocapture

Setting capture to TRUE in the config object of the payment request will automatically capture the payment upon confirmation, making it ideal for intangible goods where immediate provisioning is expected. The payment capture ID will be returned in the response and should be stored for future reference.

Request example:

```

Unset
{
  "currency": "EUR",
  "payment_amount": "10000",
  "payment_authorization_reference": "9145e250-1669-42e1-b02b-cd81ef0c5c52",
  "config": {
    "capture": true
  }
}

```

Response example:

```
Unset
{
  "state_context": {
    "payment_transaction_id":
    "krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0"
  },

  "previous_state": "PENDING_CONFIRMATION",
  "payment_request_id": "krn:payment:eu1:request:81b70347-972f-65fb-95d7-b7aa036aca7e",
  "state": "AUTHORIZED",

  "payment_capture_id":
  "krn:payment:eu1:transaction:bc42b6ff-b222-463c-b4b2-d8d6a82e0162:capture:1",
  "state_expires_at": "2024-07-21T12:43:11.113998338Z",
  "expires_at": "2024-07-21T12:43:11.113998338Z",
  "created_at": "2024-07-19T12:43:11.113998338Z",
  "updated_at": "2024-07-19T12:43:11.113998338Z"
}
```

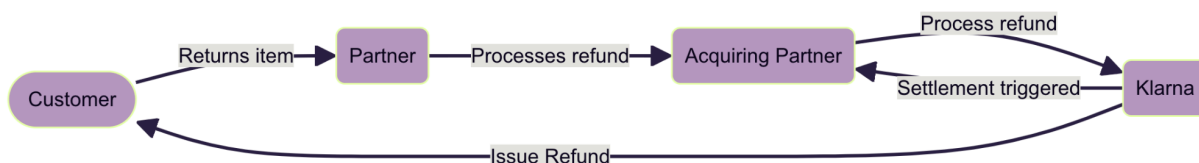
2.8.3 Refund payment transactions and allocation

There are two options available for processing refunds for a transaction that support different integration patterns used by partners. Each of these options has specific requirements and recommendations that enable optimal integration to provide the best post-purchase experience for customers.

⚠ Klarna supports a maximum of 200 refunds and 500 total operations on any given transaction. Exceeding this restriction will result in a 403 response

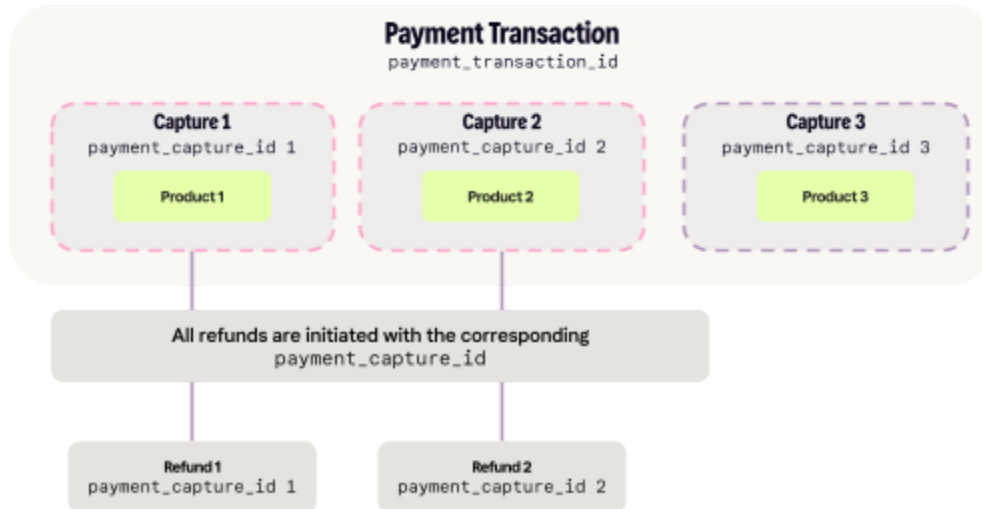
Operation	Requirement	Notes
Refund Payment Capture	Retain the payment_capture_id	This operation simplifies integration for partners that do not have line_items in their integration. Retaining the payment_capture_id and sending helps Klarna to allocate refund amounts to correct captures.
Refund Payment Transaction	Include line_items in the request	This operation requires including line_items in the request and is necessary for Klarna to be able to do proper refund allocation.

The effects of refunds on all involved parties are illustrated in the diagram below:



2.8.3.1 Refund payment capture

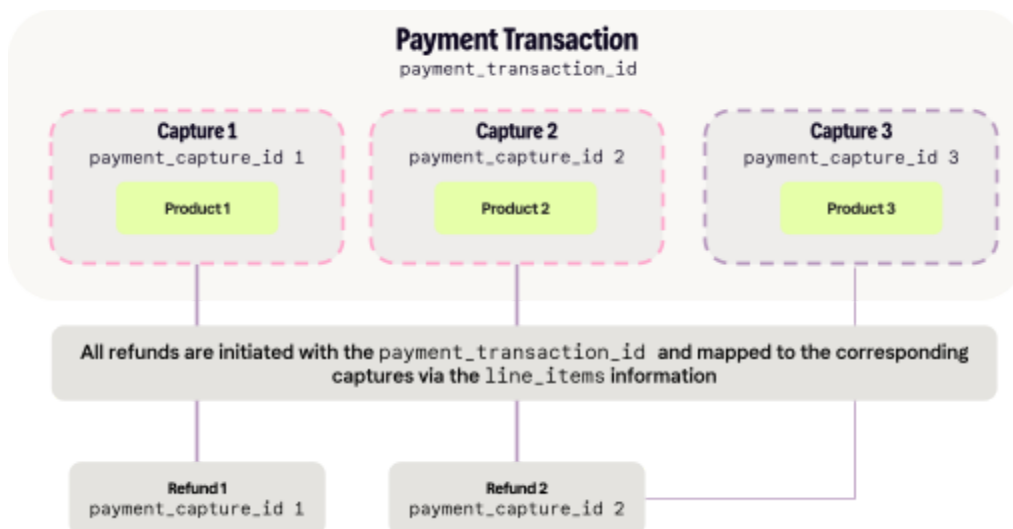
This operation is used to initiate the refund process when customers return items they've purchased. Refunds in this case are allocated to the correct captures through the `payment_capture_id` that is sent to initiate the operation.



Consult the [API reference](#) for a complete description of the request body parameters.

2.8.3.2 Refund payment transaction

In this scenario refunds are allocated to the correct captures through the `payment_transaction_id` and `line_items` that are sent to initiate the operation. In the case of a transaction with multiple captures, it is crucial for Klarna to receive `line_items` details to allow proper allocation to the corresponding capture.



Consult the [API reference](#) for a complete description of the request body parameters.

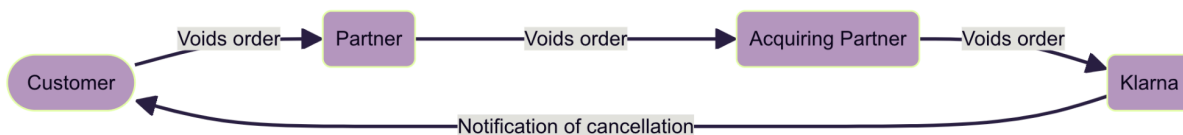
2.8.4 Void payment transaction

This operation should be performed when the payment is no longer intended to be fulfilled. For example, this can happen when customers opt to cancel a purchase or when a Partner needs to release the remaining authorization when a product is not available for shipment.

It can only be performed:

- If the transaction has not already been fully captured.
- If the transaction has expired and you do not intend to reauthorize to fulfill it

The effects of voiding a transaction on all parties involved are illustrated in the diagram below:

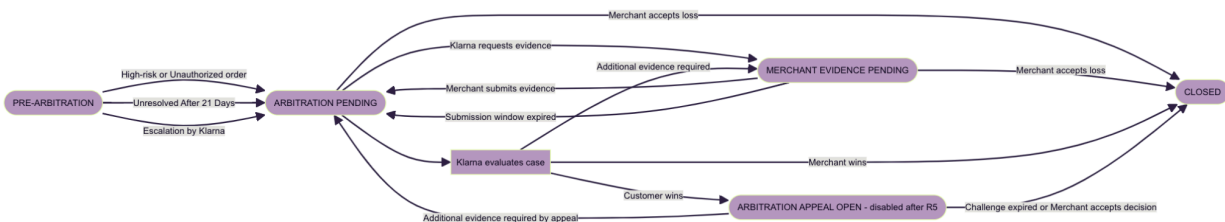


Consult the [API reference](#) for a complete description of the request body parameters.

2.9 Disputes handling

A "dispute" occurs when a customer questions a payment due to issues like not receiving goods, unauthorized transactions, or being unhappy with a product. This section explains how disputes are handled within Klarna's system. Managing disputes well is important to keep things fair, clear, and resolved quickly, which helps maintain customer trust and avoid financial losses.

By following these guidelines, businesses can protect their finances and keep strong relationships with customers. Efficiently resolving disputes reduces chargebacks and builds customer loyalty.



A dispute typically progresses through several states from initiation to resolution. Understanding these states is crucial for effectively managing disputes:

- **Pre-arbitration:** The dispute is initiated by the customer. Klarna notifies the partner, providing initial details about the issue.
- **Arbitration Pending:** If the dispute is not resolved promptly, Klarna begins an investigation. This involves gathering additional information from both the customer and the partner or reviewing the provided information to make a decision.
- **Merchant Evidence Pending:** The partner is asked to review the case and provide evidence or a response to support their position.

- **Closed:** The dispute is resolved. Depending on the outcome, the dispute can be closed in favor of the customer, possibly triggering a chargeback, or in favor of the partner, resolving the issue without further action.

Each state represents a step in the dispute resolution process, and understanding these steps helps partners respond effectively at each stage.

Alternate

For Acquiring Partners without a dispute handling flow or Partner-facing dashboard, direct provisioning of Klarna's Partner Portal may be given to Partners, allowing direct management of disputes.

Integration of Disputes webhooks by the Acquiring Partner is still recommended in these cases, to ensure the Acquiring Partner has visibility on risk factors and outstanding dispute cases.

2.9.1 Common dispute use cases

Disputes can arise in various situations during a payment transaction. Below are some scenarios to illustrate different pathways to resolve disputes. A dispute is considered resolved and closed once:

- **Loss accepted:** If the evidence clearly supports the customer's claim, the partner may choose to accept the loss and issue a refund or replacement.
- **Closed - Lost:** Evidence provided by the Partner was insufficient, resulting in Klarna ruling in favor of the customer.
- **Closed - Won:** Evidence provided by the Partner proved that they acted appropriately and all goods were delivered as promised, resulting in Klarna ruling in favor of the Partner.

Example 1: Non-receipt of goods:

Scenario: A customer named Emily places an order for a laptop from ShopStore Electronics. After the expected delivery date passed, Emily contacts ShopStore to report that she hasn't received the laptop.

Dispute Trigger: Emily initiates a dispute through Klarna, claiming non-receipt of goods.

Resolution Pathway: ShopStore reviews their shipping records and discovers that the package was delayed. They provide tracking information and proof of shipment through Klarna's Dispute API. Emily is notified of the delay, and the dispute is resolved and closed when she confirms receipt of the package.

Example 2: Damaged Goods Received:

Scenario: James orders a set of dining chairs from HomeStyle Furniture. Upon delivery, he finds that one of the chairs is damaged.

Dispute Trigger: James raises a dispute, stating that the goods were delivered in poor condition.



Resolution Pathway: HomeStyle Furniture reviews the claim and requests photo evidence of the damage. After verifying the claim, they offer James a replacement chair at no additional cost. James accepts the offer, and the dispute is closed in his favor.

Example 3: Incorrect Billing:

Scenario: Olivia purchases a smartwatch from GadgetGrove using Klarna's payment services. However, when she checks her bank statement, she notices that she has been charged twice for the same item.

Dispute Trigger: Olivia contacts GadgetGrove and subsequently raises a dispute through Klarna, claiming an incorrect charge.

Resolution Pathway: GadgetGrove reviews their payment records and confirms that a duplicate charge occurred. They promptly refund the extra payment, and the dispute is resolved and closed in Olivia's favor.

Example 4: Unauthorized Purchase:

Scenario: Michael notices a transaction on his account for a high-end gaming console from GameHub, a store he has never shopped at.

Dispute Trigger: Michael suspects fraud and files a dispute through Klarna, stating that the purchase was unauthorized.

Resolution Pathway: GameHub reviews the transaction and discovers that it was made with stolen card information. They cancel the order and refund Michael, resulting in the dispute being closed. Klarna also investigates and assists in securing Michael's account to prevent future fraudulent activity.

Example 5: Goods Not as Described:

Scenario: Sophia orders a designer handbag from ChicStyles, expecting the product advertised online. However, when it arrives, the bag is of a different color and material than described.

Dispute Trigger: Sophia initiates a dispute, claiming the product she received does not match the description.

Resolution Pathway: ChicStyles reviews the product listing and the photos provided by Sophia. They acknowledge the error and offer a full refund or exchange for the correct item. Sophia chooses the exchange, and the dispute is resolved and closed.

2.9.1.1 Accept Loss

In certain situations, it may be more practical for a partner to accept a loss rather than contest a dispute. Accepting a loss means acknowledging that the customer's claim is valid and agreeing to resolve the dispute by issuing a refund or replacement without further investigation. This approach



can save time and resources, especially when the evidence strongly supports the customer or when the cost of defending the dispute outweighs the potential loss.

Consider accepting a loss in the following scenarios:

- **Strong Evidence Against the Partner:** The evidence provided by the customer is compelling, such as clear proof of non-receipt, unauthorized transactions, or defective products.
- **High Cost of Defense:** The cost of gathering and submitting additional evidence, such as legal fees or operational costs, exceeds the amount in dispute.
- **Customer Relationship Considerations:** Maintaining a positive relationship with the customer may be more valuable than the disputed amount, especially in cases involving loyal customers or high-value transactions.
- **Small Disputed Amounts:** When the amount in dispute is relatively small, it may not be worth the effort to contest the dispute.

Accepting a loss can have several implications:

- **Financial Adjustments:** The disputed amount will be refunded to the customer, and any associated fees, such as chargeback fees, will be applied to the partner.
- **Impact on Payouts:** Accepting a loss may result in adjustments to your payouts, depending on the payment method and terms of the dispute.
- **Customer Trust:** Resolving the dispute quickly and amicably by accepting a loss can enhance customer satisfaction and loyalty.

The process for accepting a loss typically involves the following steps:

1. **Review the Dispute:** Assess the evidence provided by the customer and determine if accepting the loss is the best course of action.
2. **Use Klarna's Dispute API:** To accept the loss, make the appropriate API call through Klarna's Dispute API. This action will close the dispute and trigger the refund process.
3. **Document the Decision:** Keep a record of the decision to accept the loss, including any evidence reviewed and the reasoning behind the decision. This documentation can be valuable for future reference or internal audits.
4. **Communicate with the Customer:** Notify the customer that their dispute has been resolved in their favor. This communication should be clear, concise, and appreciative, reinforcing a positive customer experience.

Example:

FashionFusion, an online store, receives a dispute from Karen, a customer claiming that the handbag she received was damaged. Karen provides clear photos showing the damage, and the cost of shipping a replacement is low compared to the potential cost of contesting the dispute. After reviewing the evidence, FashionFusion decides to accept the loss and promptly issues a refund to Karen. This resolution not only saves time and resources for FashionFusion but also leaves Karen satisfied with the quick and fair resolution.

2.9.1.2 Upload Shipping/Delivery Evidence

Providing shipping or delivery evidence is crucial in resolving disputes where a customer claims non-receipt of goods or delayed delivery. By submitting clear and accurate evidence, you can substantiate that the goods were shipped or delivered as agreed. This evidence plays a vital role in defending against disputes and ensuring that the resolution process is fair and transparent.

Importance of Shipping/Delivery Evidence: In disputes involving non-receipt or delayed delivery of goods, shipping or delivery evidence is often the deciding factor. This evidence helps verify that the transaction was completed according to the terms agreed upon by both the partner and the customer. Without this evidence, the dispute is likely to be resolved in favor of the customer, potentially leading to financial loss and a negative impact on your business's reputation.

You can submit various types of evidence to support your case, including:

- **Shipping Receipts:** Documentation that confirms the goods were shipped to the customer's address.
- **Tracking Numbers:** Unique identifiers provided by the shipping carrier that allow tracking of the shipment's progress.
- **Delivery Confirmations:** Proof from the shipping carrier that the goods were delivered to the specified address. This may include a signature or electronic confirmation.
- **Shipping Labels:** Images or scans of the shipping label that was attached to the package, showing the destination address and other relevant details.

To effectively upload and submit shipping or delivery evidence, follow these steps:

1. **Gather the Necessary Documents:** Collect all relevant documents related to the shipment, such as tracking numbers, shipping receipts, and delivery confirmations.
2. **Use Klarna's Dispute API:** Access Klarna's Dispute API to upload the evidence. The API allows you to attach files and submit them as part of your response to the dispute. Ensure that the documents are clearly labeled and easy to identify.
3. **Verify the Information:** Before submitting, double-check the evidence to ensure it is accurate, complete, and directly related to the disputed transaction. Incorrect or incomplete evidence may weaken your case.
4. **Submit the Evidence:** Use the API to upload and submit the evidence. Once submitted, Klarna will review the documentation as part of the dispute resolution process.
5. **Track the Dispute Status:** After submitting the evidence, monitor the status of the dispute through Klarna's webhooks or API updates. This will help you stay informed about any further actions required.
6. **Maintain Records:** Keep copies of all submitted evidence for your records. This can be useful for future disputes or internal audits.

Example:

UrbanGear, an online store, receives a dispute from Chris, a customer claiming that the hiking boots he ordered were never delivered. UrbanGear gathers the shipping receipt, which includes the tracking number and delivery confirmation from the carrier, showing that the boots were delivered to Chris's address. They use Klarna's Dispute API to upload these documents as



evidence. After reviewing the submission, Klarna resolves the dispute in UrbanGear's favor, as the evidence clearly supports that the goods were delivered as promised.

2.9.2 Managing Disputes

Managing disputes effectively requires a systematic approach to ensure timely and accurate responses. Below are the key steps to manage disputes using Klarna's tools and resources.

Step 1: Listen to Dispute Webhooks to Receive Proactive Dispute Communications

Klarna webhooks allow you to receive real-time notifications about dispute activities. This proactive communication enables you to respond swiftly to any dispute-related events, minimizing the risk of delayed responses or escalations. As soon as you receive a notification, evaluate the information and prepare to take the necessary actions, whether it's gathering evidence, reviewing the dispute, or preparing a response.

To stay informed about dispute progress, utilize the following webhooks:

- `payment.dispute.state-change`: Tracks transitions between dispute states such as Pre-Arbitration, Arbitration Pending, and Closed.
- `payment.dispute.updated`: Notifies when dispute details or evidence have been updated, helping partners ensure timely responses and evidence submissions.

More information on setting up webhooks is available in [Subscribing to webhook events](#).

Enablement of webhooks allows for:

- Immediate awareness of dispute events.
- Reduced response times.
- Proactive management of disputes before they escalate.

Step 2: Display Dispute Updates to the Partners

Keeping partners informed of the latest dispute updates is crucial for transparency and effective dispute resolution. Displaying this information clearly and promptly allows partners to take the necessary actions without delay.

- **Retrieve Dispute Updates:** Use Klarna's Dispute API to fetch the latest updates on ongoing disputes. This includes changes in dispute status, requests for additional evidence, and any decisions made by Klarna.
- **Display Updates Clearly:** Integrate the dispute updates into your existing partner dashboard or management system. Ensure that the information is presented in a clear, concise format that partners can easily understand and act upon.
- **Highlight Actionable Items:** Clearly mark any updates that require immediate attention or action from the partners, such as requests for evidence or deadlines for responses.

Benefits:

- Partners are always informed of the current status of disputes.



- Clear communication of actionable items reduces the risk of missed deadlines or incomplete responses.
- Enhanced transparency and collaboration between you and your partners.

Step 3: Allow Partners to Respond to Disputes via Your Existing Disputes Infrastructure

Enabling partners to respond to disputes directly through their existing systems streamlines the dispute management process. This approach ensures that responses are timely and that all required evidence and information are submitted efficiently.

- **Integrate Response Capabilities:** Ensure your dispute management infrastructure is integrated with Klarna's Dispute API, allowing partners to submit their responses directly through your system.
- **Submit Evidence and Responses:** Partners should be able to upload relevant evidence—such as shipping receipts, tracking numbers, and customer communications—directly through your system. This information is then transmitted to Klarna for review.
- **Monitor Submission Status:** Keep track of the status of submissions to ensure all required documentation has been successfully uploaded and received by Klarna.



2.10 Pricing and reconciliation

2.10.1 How pricing works

When a transaction is confirmed, Klarna issues rate details specific to the transaction in what is referred to as a "promise". These rate details appear in the `confirm_payment` request API response and are 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.

It is important not to share the rate provided with Partners as a final amount since it is subject to adjustments based on the actual captured amount and quantity, and it reflects 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/payment/requests/{payment_request_id}](#). To view the broader rates across their pricing plan, you should use the GET read a price plan request. This ensures accurate and up-to-date rate application in all transactions and calculations.

Release notes

NEW
17 GET read a price plan request will be supported in future releases.

2.10.1.1 Definitions and Calculations

Effective 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, sales channel (physical or online store), and payment program. The fixed and variable costs shared on confirmation are applied to a transaction on a per capture basis.

Discounts: are adjustments applied to a given transaction based on certain criteria or promotions. Discounts reduce transaction fees based on certain criteria or promotions. The effective rate communicated accounts for any applied discounts.

Rate communication: occurs 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

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

2.10.2 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.

2.10.2.1 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.

2.10.2.2 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.

2.10.2.3 Handling adjustments

The settlement file will detail captures, fees, refunds, disputes, and discounts. 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 customer), and a fee may be applied. More details are available in the [disputes](#) section.
- **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 and must be passed on to Partners in a 1:1 ratio.

2.10.2.4 Consuming rate and fee data

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

- **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. If a microtransaction cap applies to the transaction, 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.

2.10.2.5 Communicating costs to Partners

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

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

2.10.3 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 customer and captured/activated the payment.	2018-08-10T07:45:00Z



Parameter	Definition	Example
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.	krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0
klarna_reference	The reference to identify the customer-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
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 Partner 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



Parameter	Definition	Example
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 Partner	krn:partner:global:account:live:LWT2XJSE
settlement_id	Unique identifier for the settlement and payout, will be the reference on the bank statement	

2.10.3.1 Settlement webhooks

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

Use case	When	Event name
Payout processed	When the settlement has resulted in a payout, but has not yet been run. The payout request has been sent from Klarna's internal system.	settlement.payout.processed
Payout confirmed	Klarna has successfully run the payout. This does not guarantee the availability of funds in the receiving bank account.	settlement.payout.state-change.confirmed
Payout delayed	When a payout is delayed because of Klarna internal issues.	settlement.payout.state-change.delayed
Payout failed	When a payout has failed, either when initiating payout or when the receiving bank rejected it.	settlement.payout.state-change.failed
Settlement Report generation delayed	Report generation delayed for internal reasons. This is independent of the actual payout.	settlement.payout.report.delayed
Settlement	When the settlement data from Klarna's	settlement.payout.report.created



Report available on API	internal system. has been imported and is available in any format in the API.	
Settlement Report sent to SFTP	When the settlement reports have been uploaded to the SFTP	settlement.payout.report.uploaded-to-sftp
New invoice available	When an invoice has been created	settlement.invoice.created
Bank account added	When a bank account is added to Klarna's internal system.	settlement.configuration.bank-account.added
Bank account removed	When a bank account is added to Klarna's internal system	settlement.configuration.bank-account.removed
Bank account update	When a bank account is added to Klarna's internal system	settlement.configuration.bank-account.updated

2.10.3.1.1 Payload structure per use case

The structure of the payload will vary based on the event type however the metadata will always follow the structure documented in the [Configure Klarna Webhook](#) section.

Event name	Payload structure
settlement.payout.processed	"settlement_id"
settlement.payout.state-change.confirmed	"amount"
settlement.payout.state-change.delayed	"currency"
settlement.payout.state-change.failed	"settling_business_entity_id"
settlement.payout.report.delayed	"settling_business_entity_name"
settlement.payout.report.created	"settlement_configuration_id"
settlement.payout.report.uploaded-to-sftp	"payout_date"
settlement.invoice.created	"bank_account_details": <ul style="list-style-type: none"> • "bank_account_id" • "routing_number" • "account_number" • "account_holder"
settlement.configuration.bank-account.added	"amount"
settlement.configuration.bank-account.removed	"currency"
settlement.configuration.bank-account.added	"settling_business_entity_id"
settlement.configuration.bank-account.added	"bank_account_id"
settlement.configuration.bank-account.added	"bic"
settlement.configuration.bank-account.added	"account_number"
settlement.configuration.bank-account.added	"account_holder"
settlement.configuration.bank-account.removed	"settling_business_entity_id"
settlement.configuration.bank-account.removed	"bank_account_id"
settlement.configuration.bank-account.removed	"bic"
settlement.configuration.bank-account.removed	"account_number"
settlement.configuration.bank-account.removed	"account_holder"

settlement.configuration.bank-account.updated	<pre> "settling_business_entity_id" "bank_account_id" "updated_at" "previous_fields" • "bank_account_id" • "bic" • "account_number" • "account_holder" "updated_fields" • "bank_account_id" • "bic" • "account_number" • "account_holder" </pre>
-----------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Payload example

The following example reflects the payload structure for settlement.payout.processed event:

```

Unset
{
  "metadata": { ... },
  "payload": {
    "settlement_id": "6482846287",
    "amount": 12500,
    "currency": "USD",
    "settling_business_entity_id": "K123456",
    "settling_business_entity_name": "Shoes and Hats Inc.",
    "settlement_configuration_id":
"krn:partner:global:settlement:settlement-config:ad71bc48-8a07-4919-a2c1",
    "payout_date": "2024-08-03",
    "bank_account_details": {
      "bank_account_id":
"krn:partner:global:settlement:bank-account:0fb1cd5f-e3d5-4e2d-847f",
      "routing_number": "021000021",
      "account_number": "473294629",
      "account_holder": "Hats and Shoes Inc."
    }
  }
}

```

2.10.4 Cut-off times for different regions

Cut-off times define which payments will be included in a given day's settlements. All payments which have been captured or refunded before the cut-off time will be added to the settlement for that period and a payout will be initiated by Klarna on the next banking day. The settlement file will be provided within 14 hours of the cut-off time.

The **Payout Date** is when Klarna initiates the payout to the partner and publishes the settlement report. The actual arrival of the payout in the partner's bank account is dependent on their bank's processing times.

Region	Klarna Cut-off time
Europe (including Great Britain)	00:00 GMT (UTC+1)
Asia Pacific	00:00 AEST (UTC +10:00)
North America	00:00 EST (UTC -5)



2.11 Test your integration

Klarna's test environment is designed for you to test your integration without incurring actual charges or moving real funds. This simulation allows you to verify that your integration with Klarna's systems functions correctly. Using the test mode is crucial for detecting and resolving potential issues. It should be an integral part of all release procedures to ensure robust testing before deployment to production.

Klarna mandates that all integrators undergo thorough testing in the test environment and provide Klarna comprehensive access to their testing setup for additional validation before going live.

Testing within the live environment is generally discouraged due to the potential for rejections that naturally occur during Klarna validations. However, if testing in the live environment is necessary, be aware of common reasons for rejections:

- 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

To minimize complications, focus on extensive testing within Klarna's test environment and limit live tests unless absolutely necessary.



3. Test cases

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

3.1 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.

3.1.1.1 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.

3.1.1.2 Test case 2: Onboard a new Partner

Steps to follow:

1. Onboard a new Partner 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.

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

Klarna 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 Partner and get the credentials.
2. Suspend a payment product.
3. Revert the suspension.

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

Channels represent how Klarna's products are shown to the end customers, 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 customers, enhancing their experience.


Release notes

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

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

Channel collection information is crucial for enhancing the customer'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 Partner 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.*

3.1.1.6 Test case 6: Fetch and update account information

It's important that the account information reflects the latest information about your Partners 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.

3.1.1.7 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.

3.1.1.8 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.

3.1.1.9 Test case 9: Error handling

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




3.2 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 customer 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 customer data and sample payment data provided [here](#).*

3.2.1 Online store end-to-end test cases

3.2.1.1 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 customer details match the information provided during the customer flow.
2. Process a full capture: simulate shipping the goods to the customer, 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.*

3.2.1.2 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 customer, 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

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

3.2.1.3 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 customer, 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 customer, 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



¹⁷ *Payment transaction management API will be available via the Klarna Network APIs in future releases*

3.2.1.4 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

¹⁷ *Payment transaction management API will be available via the Klarna Network APIs in future releases*

3.2.1.5 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 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*

3.2.1.6 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*

3.2.1.7 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*

3.2.1.8 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 customer 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 customer can change the payment method after the initial failed payment attempt.

3.2.1.9 Test case 9: Complete a free purchase transaction

Steps to follow:

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

Expected outcome:

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

3.2.1.10 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.

3.2.2 Mobile app test cases

3.2.2.1 Test case 1: "Remember me" function per app

Steps to follow:

1. Launch the app and navigate to the login screen.
2. Enter valid login credentials.



3. Enable the "Remember me" checkbox before logging in.
4. Complete the login process.
5. Log out from the app.
6. Close and reopen the app, navigating back to the login screen.

Expected outcome:

- Customer's login credentials should be pre-filled, indicating the "Remember me" function works correctly within an individual app.
- The customer should be logged in faster if auto-login is supported.

3.2.2.2 Test case 2: "Remember me" feature across different apps

Steps to follow:

1. Log into one app with the "Remember me" option enabled.
2. Close the first app and open a second app integrated with SSO.
3. Navigate to the login screen or a function requiring one's personal details.

Expected outcome:

- The customer's details are pre-filled when required, validating shared login and remembered login details across different apps using SSO.
- The customer doesn't need to re-enter the login credentials if the SSO session is active.

3.2.2.3 Test case 3: Redirect to browsers and third-party apps

Steps to follow:

1. Initiate a payment transaction that requires third-party verification (e.g. BankID in Sweden or similar).
2. Verify that the redirection to an external browser or a third-party app occurs.

Expected outcome:

- The app redirects to the intended external browser or third-party app without errors, ensuring customers can be redirected correctly for authorization within the purchase flow.

3.2.2.4 Test case 4: Redirect back from third-party apps

Steps to follow:

1. From the Partner app, initiate a flow that requires authentication via a third-party app.
2. Complete the required actions within the third-party app.
3. Verify if the redirect back to the Partner app happens after completion.

Expected outcome:

- The customer is smoothly redirected back to the Partner app to proceed with or complete the transaction, after interaction with third-party applications.

3.2.2.5 Test case 5: US Bank secure web limitation

Steps to follow:

1. Initiate a transaction that requires redirecting to a US bank's web page known to enforce secure web limitations.
2. Observe the method by which the Mobile SDK handles this redirect.

Expected outcome:

- Mobile SDK renders the bank's page in a secure web window successfully, without using in-app webviews, addressing the secure web limitations imposed by US banks.

3.2.2.6 Test case 6: Onfido (ID Scan) Know Your Customer flow

Steps to follow:

1. During the payment transaction, initiate the KYC flow.
2. Allow camera access and follow instructions to scan an ID document.
3. Submit the ID document for verification.

Expected outcome:

- The app effectively scans the ID with the camera and completes the verification process, confirming the customer's identity.

3.2.2.7 Test case 8: 3DS card verification

Steps to follow:

1. Initiate a payment transaction using a credit card requiring 3DS verification.
2. Verify that the redirect to the bank's 3DS page is successful.
3. Complete the 3DS verification process.

Expected outcome:

- The payment transaction pauses for 3DS verification and upon successful verification, the transaction continues and completes successfully, ensuring the 3DS verification process for card payments functions correctly.

3.2.2.8 Test case 9: Klarna app Handover

Steps to follow:

1. Initiate a payment transaction with Klarna app installed.
2. Verify that the handover to the Klarna app occurs automatically.
3. Complete necessary authentication or consent in the Klarna app.
4. Check for a redirect back to the original app after completion.

Expected outcome:

- The Klarna app opens without issue, presenting the required authentication or consent screens, followed by a redirect back to the original app to continue or complete the

transaction, indicating a seamless app handover for authentication and consent within the Klarna app.

3.2.3 Customer token end-to-end test cases:

3.2.3.1 Test case 1: Present Klarna as a subscription payment method and retrieve a customer token

Steps to follow:

1. Create a payment request with additional tokenization parameters
2. Complete the payment request
3. Once the payment request is confirmed using the payment confirmation token, retrieve the customer token from the response body.

Expected outcome:

- Klarna is presented as a payment method for the subscription.
- A customer token is successfully created and stored, and can be used for future charges.

3.2.3.2 Test case 2: Subscription based token charge

Steps to follow:

1. Create a customer token with the "Customer Not Present on Payment" scope.
2. Charge a tokenized customer for a subscription payment.

Expected outcome:

- The customer is charged with the correct amount.
- The subscription and line item details are included in the token charge call.

3.2.3.3 Test case 3: On-demand based token charge

Steps to follow:

1. Create a customer token with the "Customer Present on Payment" scope.
2. Login to the partner system as the customer and initiate a transaction
3. Handle step-up authentication flow. Forward the customer to the returned distribution URL and confirm the payment request when the customer has completed the Klarna Payment flow.

Expected outcome:

- The customer is authenticated with a customer account, and can complete transactions without further customer interaction, enabling one-click payments and faster checkouts.
- The customer is redirected to step-up authentication flow when necessary.
- The correct amount, subscription and line item details are included in the token charge call.

3.2.3.4 Test case 4: Mixed payments token charge

Steps to follow:



1. Combine baskets that include both an untokenized transaction and setting up tokenization for recurrent transactions for later payments.
2. Complete the payment request
3. Once the payment request is confirmed using the payment confirmation token, retrieve the customer token from the response body.

Expected outcome:

- The initial one-time purchase is processed, and subsequent subscription charges are successfully made using the customer token.
- The correct amount, subscription and line item details are included in the token charge call.



3.3 Sample customer 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 test environment.

3.3.1 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.
- Partner account data: please use the following information to test [Partner onboarding](#).

3.3.1.1 Account owner

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

3.3.1.2 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). Data in the test environment is not treated as real PII.

3.3.2 Sample customer data

To test the standard approved and denied payment flows in Klarna Payment Services use this [data](#).

3.3.3 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 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
Belgium	Min: 1 EUR Max: 1500	NA	NA	Min: 0 EUR Max: 10000	Min: Max:	Min: 0.1 EUR Max: 14000

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
	EUR			EUR		EUR
Canada	NA	Min: 35 CAD Max: 1500 CAD	NA	Min: 0 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 NZD	NA	NA	NA	NA
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	Min: 0 NOK Max: 100000 NOK	NA	NA



Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
			2400-150000 NOK 24 months 3600-150000 NOK			
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
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



Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
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



3.4 Create public documentation for your Partners

Release notes

¹⁷ Additional guidance on how to create public documentation for your Partners, and how to present Klarna will follow in subsequent releases.

3.5 Integration checklist - Go-live

Before you go live, use the following checklists to guarantee your integration with Klarna is thorough and you're ready to enable Klarna Payment Services in production. This ensures you're following best practices and guidelines we've shared for a seamless adoption.

¹⁷ In future releases, this section will encompass both use cases:

- Checklist for Acquiring partners
- Checklist for Partner integrating Klarna directly

3.5.1 Account - Management API

Partner account credential configuration

- Client-side and Server-side credentials can be created, updated, and deleted for both Test and Live environments. More information in [Account credential management](#).

Onboard and manage Partners

- Klarna account structures aligned to internal account structures to minimize friction in accordance with [Step 1: Determine account structure](#)
- Partners can be onboarded to Klarna with all available information, and the data has been confirmed in line with [Step 2: Onboard Partners](#)
- All interactions follow the Klarna guidelines for [Integration resilience](#).
- All onboarded accounts are granted access to the Partner portal via deeplinking or direct access as described in [Step 1: Grant access to Klarna's ecosystem](#)

Manage your merchant accounts and payment products

- All updates to account data provided within Partner systems are propagated out to Klarna without any manual intervention as described in [Onboard and manage Partners](#), and this behavior has been confirmed.
- Merchant payment products and account statuses are kept aligned with their status within partner systems via the mechanisms described in [Step 3: Manage your Partner payment products](#) and [Account suspension](#).
- Channel types are defined and managed in line with the Partners expectations given the behavior of other payment methods as outlined in [Channel types and management](#)
 - Webhooks are fully integrated, and redundancies put in place to allow graceful failure if incidents occur as outlined in [Subscribing to webhook events](#) and [Integration resilience](#).

Complete test cases details in the [Test your integration](#) section.



3.5.2 Partner product API

Design your Klarna solution

- Considerations and requirements for when and how to initiate the payment process, present Klarna, and optimize business results. Learn more [here](#).
- [ensure the UX is optimized](#) for conversion.
- Configure webhooks
 - Monitor payment states and retrieve payment confirmation tokens using [notification webhooks](#).

Process payments using Klarna Payment Services

- Learn how to process payments using Klarna Payment Services from one time payment in [online stores](#), [physical stores](#), [subscriptions](#) or [On-demand](#).
- Ensure all interactions follow the Klarna guidelines for [Integration resilience](#).

Manage Klarna payment transaction

- Capture, update, refund and void payment transactions. See more details [here](#).

Manage disputes

- Subscribe to dispute webhooks to receive proactive notifications, display updates to your Partners, and allow them to respond to disputes via your existing disputes infrastructure. See more details in [Disputes handling](#).


Manage settlements

- Configure and subscribe to settlement webhooks to get immediate notification of settlement related events. See more details [here](#).

Complete test cases details in the [“Test your integration”](#) section.

3.5.3 UX - Customer flow

Release notes

 Customer flow UX checklist will be made available in a future release.

3.5.4 Reflect your partnership with Klarna

Ensure you have completed all the points mentioned in the [Create public documentation for Partners](#) section.

3.5.5 Set up your production environment

We have designed our live and test environments to function similarly. Switching environments primarily involves swapping your API keys.

For account management:



- Get your production [account credentials](#) and switch them.
- Set up [webhooks](#) in your production environment.
- Ensure you are not relying on any test paths.

For payments:

- Get access to your production environment in the Partner portal.
- Get your production API credentials and switch them.
- Set up [webhooks](#) in your production environment.
- Ensure you are not relying on any test paths.



4. Resources

4.1 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 [Retrieval of Descriptors](#). This approach allows to tailor the customer 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 customer expectations and industry developments. Stay updated with the latest features from Klarna to further improve the customers experience.

Further recommendations are shared throughout this documentation. By adopting a customers-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 customers.

4.2 Klarna ecosystem

4.2.1 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 customer'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 payment 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 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.

4.2.1.1 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

4.2.1.2 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

4.2.2 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

4.2.2.1 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.

4.2.2.2 Ensuring a high-quality integration

To optimize your integration and prepare for future updates, ensure you are following Klarna's recommendations:

- **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 customer 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.

4.2.3 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 product suite.

4.2.3.1 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.

4.2.3.2 Availability

The availability of a service indicates that it is error-free and fully usable and functional.

It is directly associated with the number of requests processed correctly and that results in responses received with status code being 2xx, 3xx and 4xx in a given month. Availability data does not include cases if, for some functional reason, there is no traffic.

In summary it is calculated as follows:

$$A \% = [SR / R] * 100$$

where:

- *A* means the availability in % during a given month.
- *SR* means the total number of successful requests responded with 2xx, 3xx and 4xx during a given month.
- *R* means total number of requests performed during a given month

4.2.3.3 Downtime

Downtimes in Klarna's payment 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 Klarna Partners, Third Party Payment Option Providers, banks or other external data providers.
- Unavailability caused by or attributable to the Klarna partner and/or any of the Partners contractors, suppliers or any other third party that the Partner 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.

4.2.3.4 Technical Support

Customer support for Partners 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](#).



4.2.4 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's SDK, see [Recommended integration: Klarna.js](#) section.

Consult the [SDK reference](#) for a complete description of the specifications.

4.2.5 Mobile SDK

Klarna Mobile SDK enables Klarna services to work seamlessly within a native mobile app.

The Mobile SDK supports Klarna Payment Services, 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 customer details, making it easier for returning customers.



4.3 Security

4.3.1 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](#).

4.3.1.1 Global authentication for Partner product API

For Klarna Partners working with multiple `account_ids`, you should consider the following:

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

4.3.2 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.

4.3.3 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.

4.3.4 Mutual Transport Layer Security

Mutual Transport Layer Security (mTLS) is an enhanced communication security mechanism that adds extra layers of protection to make it more difficult for attackers to misuse leaked credentials. It ensures that both the client and server authenticate each other, making the communication more secure.

Klarna requires acquiring partners to implement mTLS to ensure the security of Partner integrations. For all other account types, it may be enabled manually by Klarna on request.

4.3.4.1 To enable mTLS for your Klarna account, follow these steps:

1. Reach out to the Klarna partner account team to enable mTLS for your account.
2. Create and Store Your Private Key using elliptic curve cryptography with the prime256v1 specification.
 - Example command:

Unset

```
openssl ecparam -genkey -name prime256v1 -out my-key-file.pem
```

3. Create a Certificate Signing Request (CSR):
 - The Common Name (CN) in the CSR should be the last part of your account ID.
 - For example, if your account ID is `krn:partner:global:account:live:LYABCDEI`, the CN should be `LYABCDEI`.
 - Example command:

Unset

```
openssl req -new -key my-key-file.pem -out csr.pem -subj "/CN=LYABCDEI"
```

4. Send the CSR to the Klarna account team. Klarna will return the certificate via Yopass (a secure sharing service).
5. Use the issued certificate and private key to establish connections to the Klarna API.

4.3.4.2 Important Notes

- **CSR Requirements:** The CSR should only include the CN and no extra attributes.
- **Certificate Validity:** The issued certificate is valid for 3 years. You need to monitor its expiration and initiate the renewal process in advance.
- **Multiple Certificates:** You can have up to 10 active certificates at a time.
- **Revoking Certificates:** Before revoking an active certificate, ensure a new certificate is issued and installed. Use the partner API to revoke the certificate.

4.3.4.3 Certificate Rotation Process

1. Issue a New Certificate following the [same steps](#) to create and submit a CSR.
2. Install and Test the New Certificate, ensuring the new certificate works correctly.



3. Revoke the Old Certificate, using the partner API to revoke the old certificate after confirming the new one is functioning.

By following these steps, you can ensure a secure and smooth setup of mTLS for your Klarna account.

4.3.5 Security protocols and recommendations

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 customers.
- 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 Partner 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.

4.3.6 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_e1ZGI1B5dHBIRWcjZrN1dnbEVj[...]uefnc3  
  
API Key Structure:  
klarna_<live|test>_<api>_<random>  
  
API Key Example:  
klarna_live_api_e1ZGI1B5dHBIRW1tRjF5cjZrN1dnbEVjKnIqeC[...]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



4.4 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 product suite.

4.4.1 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.

4.4.2 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 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 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 test		
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

4.4.3 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.

4.4.3.1 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.

```
Unset
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

4.4.3.2 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.

4.4.4 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 Klarna’s requirements.

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





4.5 Integration resilience

4.5.1 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 customer 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.

4.5.2 Tagging

Release notes

Key
17 Integration tagging is currently under development and is subject to change. It will become available in future releases.

4.5.3 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 customers 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 used 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 customer experience.

- All interactions with Klarna are tagged with the appropriate integration details and versions as defined in [Integration tagging](#)

4.5.4 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 customers shopping, 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/...."
    }
  ]
}
```

4.5.4.1 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 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 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.



6. Annex

6.1 Changelog

- New sections:

Section	Title
2.8	Managing payments transactions
2.6.2	Tokenized payments

- Terms update:

Version 1	Current version
Merchant	Partner

6.2 Credential Creation

Different scenarios requiring creation of credentials:

Purpose	Credential type	User	Description
On-site messaging	Client ID	merchant	Merchant implementing client-side Boost products to improve conversion.
Express checkout	Client ID	merchant	
Sign in with Klarna	Client ID	merchant	
Supplementary data APIs	API key	merchant	Merchant provides additional data outside of Partner integration to improve acceptance and customer experience.
Merchant-owned client-side integration	Client ID	merchant	Merchant owns the client-side component of the integration.
Mobile app integration	API key	merchant	Merchants own elements of their mobile app integration and require additional credentials.
	Client ID	merchant	
Payment transaction management	API key	merchant	Merchant has separate inventory management integrations and wishes to automate validation of inventory.

Purpose	Credential type	User	Description
Descriptor API		merchant	Allow them to build the checkout themselves.

