



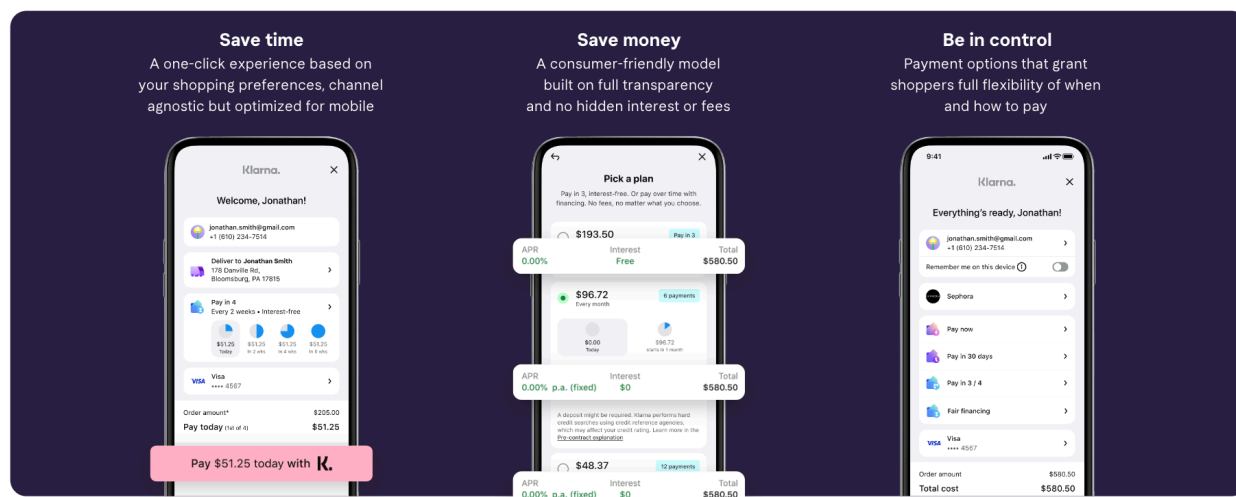
Klarna

Klarna Network Integration Guidelines

Version 2.0 - Dec 9, 2024

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 are committed to providing a seamless and secure shopping experience that helps our customers.



Klarna offers 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 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?

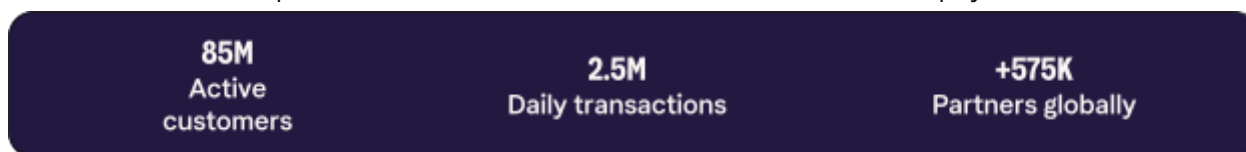
This document was last updated on Dec 6, 2024 .

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.



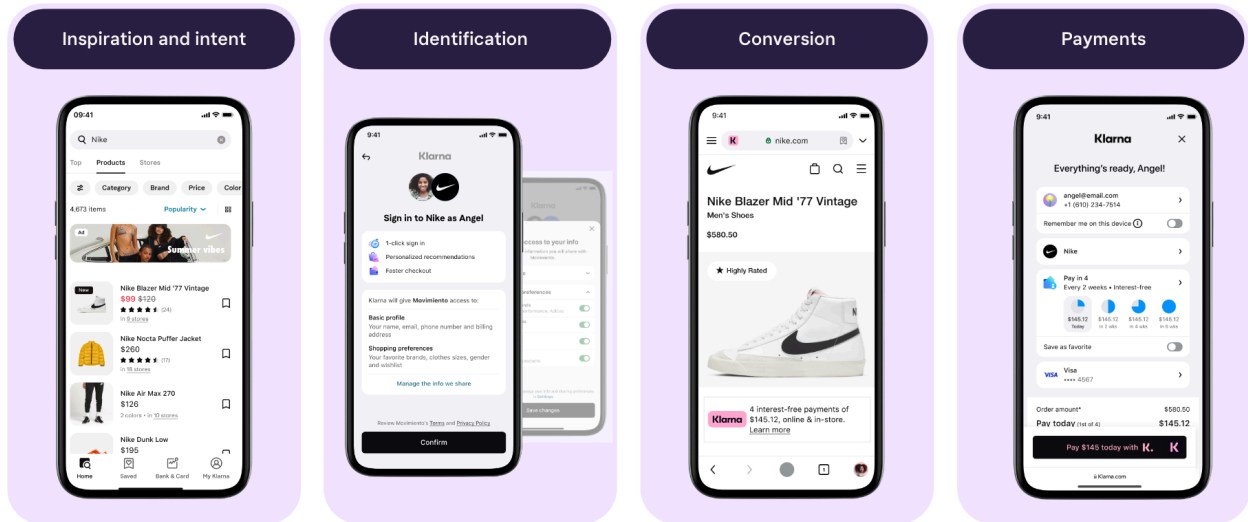
Partners grow their business with our flexible payment option and smart shopping solutions that enable customers to easily and securely pay when and how they want everywhere - online, apps and in physical stores. Klarna accommodates all shopping scenarios, from high-value transactions to everyday purchases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

- **40%** Increase in average transaction value.
- **20%** 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.

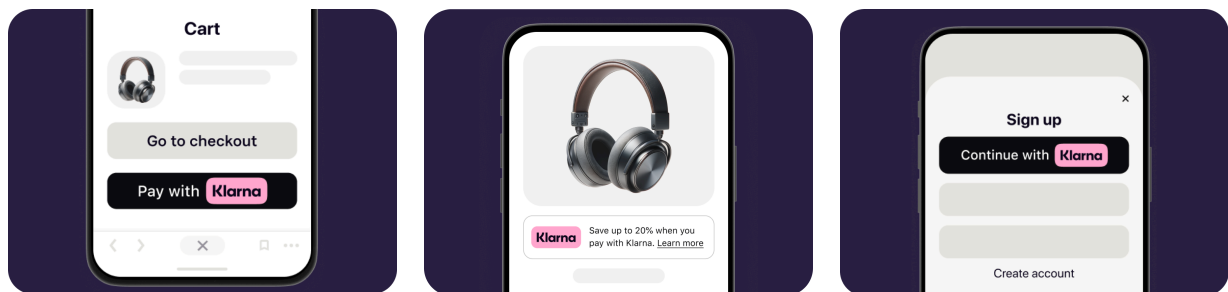
Our dynamic expertise ensures seamless interoperability between products within the Klarna ecosystem, delivering a smooth and consistent customer experience across all integration patterns.



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 Klarna's 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're 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 Getting Started

Integrate Klarna Network to offer a secure, globally accessible, and reliable payment experience. This document details required steps, recommended paths, and optional alternate flows, each marked according to their integration path and Klarna's requirements.

2.1.2.1 Prerequisites

- Klarna account setup and credentials (API keys, mTLS, etc.).
- Verification of integration path for compliance with Klarna Network Rules.



- Thoroughly review the Klarna Integration Guidelines and Network Rules to avoid delays and ensure a smooth integration process.

2.1.3 Klarna Integration Principles

Each integration must meet the following Klarna Network Requirements (KNR) principles:

- **Safe and Secure Shopping**
- **Global Availability**
- **Integration Simplicity**
- **Product Suite Interoperability**
- **High Performance**
- **Payment Programs**

More information on the Klarna Network principles is available in the Klarna Network rules..

2.1.4 Integration paths

To ensure Klarna is best able to be integrated with your solution, Klarna provides multiple integration paths which may be combined as needed to suit your architecture.

Integration Path	Description
Hosted ▾	Acquiring Partner controls client-side presentation.
API Driven ▾	Partner handles the client-side presentation.
Express ▾	Partner initiates and facilitates the flow, with the Acquiring Partner confirming payments.
Tokenized ▾	Partner recognizes authenticated users, allowing subsequent payments with limited interaction.
Platforms ... ▾	Acquiring Partners support a platform-specific integration. Where highlighted, additional requirements apply.

Multiple integration paths can coexist across different touchpoints within a single Klarna integration. Apply the most relevant category to each touchpoint, and in cases where multiple categories overlap, ensure all applicable requirements are met.

2.1.4.1 Acquiring channels:

Integration requirements may also depend on the acquiring channel being used.

Integration Path	Description
Online ▾	Covers web-based payment and account management.
Physical S... ▾	Supports in-store payment capabilities.
Mobile ▾	Enables seamless integrations for mobile apps using Klarna's mobileSDK.

2.1.4.2 Partner account management:

For non-payment services, such as brand management, dispute handling, and access to external gateways, integration requirements vary based on the availability of a Partner-facing interface.

Integration Path	Description
Dashboard ▾	Partners control the service listed within a user experience you control.
No Dashb... ▾	Your Partner experience relies entirely on external payment portals, managing each payment method individually outside of your control.

This framework provides a granular structure to clearly outline Klarna's expectations for each integration path, ensuring a consistent and reliable experience for both Partners and customers.

2.1.5 Implement Klarna Network as an Acquiring Partner

Use this to verify that each integration step meets Klarna's security, availability, interoperability, and performance standards. Integration paths are marked for specific requirements and alternate flows, each subject to Klarna approval. If only requirements are listed, this must be fulfilled.

2.1.5.1 Safe and Secure Shopping

Implement essential security measures to safeguard user data and maintain secure transactions.

- Prevent unauthorized access to Klarna APIs:* Ensure that only authorized users and systems can access your API, preventing unauthorized access.
 - **Required:** Use API keys for server-side REST APIs and Client IDs for the Web SDK. (4.3.1 API Authentication Standards).
 - **Required:** Transmit all API requests over HTTPS using TLS 1.2 or higher. (4.3.3 Communication Security).
 - **Required:** Generate and securely store signing keys for each webhook. Configure webhooks to use the designated signing key, and validate each notification. (2.4.2.2.1 Verify HMAC signature to ensure data security and integrity)
 - **Recommended:** Use unique API keys for each service or system interacting with Klarna. (4.3.6 Authentication Type by Service).
 - **Recommended:** Generate mTLS certificates according to Klarna's requirements and rotate certificates well ahead of certificate expiry. (4.3.4 Mutual Transport Layer Security).
 - **Recommended:** Rotate provided API Key when onboarded. Use the API key provided via secure link to retrieve new API credentials. Disable initial credentials. (2.4.1 Step 1: Configure account credentials)

- Maintain a high standard of data protection:* Follow (at a minimum) Klarna's recommended Security Protocols and Best Practices.
 - **Recommended:** Encrypt all stored credentials and enforce strong password policies. (4.3.5 Security Protocols and Best Practices).
 - **Recommended:** Conduct regular security audits and implement two-factor authentication for admin accounts. (4.3.5 Security Protocols and Best Practices).
 - **Recommended:** Use separate API credentials for each service to limit risk in the event of a security breach. (2.4.1.1 Account credential management)



- Efficiently support outages and other incidents:* implement and maintain a documented incident response plan that ensures prompt action and collaboration in the event of data breaches, outages, or fraud detection
- Enable Klarna to verify the reliability, accuracy, and compliance of Acquiring Partner services:*
 - [Required](#): Provide a fully functional test environment that mirrors the production setup for all integration paths, allowing Klarna to perform end-to-end testing and validation of Acquiring Partner (AP) services. (4.2.1 Environments)

2.1.5.2 Global Availability

Ensure your integration is designed for worldwide accessibility, maintaining consistent performance and service reliability.

- Display Klarna to all potential customers:* Verify the Payment status and Klarna's availability for the purchase, and present across all integrations and customer markets where eligible. The requirements and recommendations to accomplish this varies by integration path. See the list below for further detail:
 - [Required](#): Klarna Payment Services integration is built to function the same in all customer Markets: meaning payment request functions are identical in all markets without local changes. (2.6.1 Online store transaction)
 - [Required](#): Addition of new Acquiring or customer Markets to Klarna Payment Services do not incur any technical or configurational changes to Partner Integrations. (2.3.1 Checking Klarna Availability)
 - **Hosted** [Required](#): Use the `resolve()` function to determine the availability of Klarna, ensuring the customer does not proceed with Klarna unknowing that Klarna will not be available to them as a payment method. (2.6.1.1.2 Step 2: Check Klarna interoperability status)
 - **API Driven** [Required](#): Use the `read interoperability` endpoint to determine the eligibility of the Payment based on customer country and currency. (2.7.2.2.2 Shopping Session API)
 - **Platforms & Plugins** [Required](#): Use the `resolve()` function to determine the availability of Klarna, ensuring the customer does not proceed with Klarna unknowing that Klarna will not be available to them as a payment method (2.7.2.1.3 Case 3: Retrieving the Interoperability Token)
 - **Platforms & Plugins** [Alternative](#): Use the `read interoperability` endpoint to determine the eligibility of the Payment based on customer country and currency. (2.7.2.2.2 Shopping Session API)
- Ensure Klarna is presented with accurate and up-to-date messaging to all customers.*
 - **Hosted** [Required](#): Use the WebSDK to dynamically retrieve market- and language-specific descriptors. (2.3 How to Present Klarna in the Checkout).
 - **API Driven** [Recommended](#): Use the payment descriptor API to retrieve and display location-specific descriptors and payment options. (2.3 How to Present Klarna in the Checkout).

2.1.5.3 Integration Simplicity

Implement Klarna with minimal friction to ensure ease of use, efficient onboarding, and reliable operation.

- *Make adding Klarna easy for Partners:* Implement a streamlined process for onboarding Partners, ensuring complete and accurate account creation aligned with Klarna's requirements and enabling Partners to utilize Klarna services.
 - **Required:** Use Klarna's Partner Management API to automate onboarding and create new accounts efficiently (2.5 Onboard and Manage Partners).
 - **Required:** Align Klarna account structures with internal systems, ensuring consistency and completeness during the onboarding process (2.5.1 Step 1: Determine Account Structure).
 - **Required:** Collect and provide all mandatory technical and risk-related data points for each Partner during onboarding. Failure to include required data points may result in escalations or delays (2.5.2 Step 2: Onboard Partners).

- *Enable seamless system alignment* to improve fraud detection and risk assessments, while enhancing customer trust and experience through accurate brand representation and reliable account management.
 - **Required:** Automate account updates within Klarna, confirm alignment through routine checks, and implement account suspension processes to maintain accurate statuses (2.5.3 Step 3: Manage Partner Payment Products).
 - **Recommended:** Use resource-specific endpoints when making updates, allowing more robust capabilities and safer handling of resources. (2.5.3 Manage Partner payment products)
 - **Alternative:** Use the "update all subresource" endpoint for handling all configurations within an account in a single call. (2.5.3 Manage Partner payment products)
 - ⚠ This approach introduces more latency and complicates error handling (as if one resource update fails all must be re-updated).
 - **Required:** Ensure account suspension propagates between systems automatically, so that changes in account status are consistently reflected (2.5.3.2.1 Disabling a product).
 - **Required:** Integrate management webhooks to proactively manage account status changes, allowing timely updates and issue resolution (2.4.2 Configure Klarna Webhooks).

- *Deliver a purchase experience on par with direct Klarna integration:* Integrate Klarna's payment solutions to support multiple transaction types, ensuring consistent and efficient payment processing for Partners.
 - **Hosted** ▾ **Required:** Implement a client-side integration using Klarna's Web SDK to provide an interactive, real-time payment experience directly within the Partner's web environment. This enables faster checkouts and direct updates during the payment process (2.6.1.1 Payment request initiated via Klarna.mjs).
 - **API Driven** ▾ **Required:** Configure a server-side integration to handle payments securely. This approach is suited to environments where the Partner manages checkout independently, with payment confirmation and capture handled directly on the server (2.6.1.2 Server-side initiated payment request).
 - **Tokenized** ▾ **Required:** Configure subscription payments to enable recurring billing with Klarna, ensuring flexible support for subscription-based services and allowing Partners to manage automatic payments at set intervals (2.6.2.3.1 Subscriptions).
 - **Tokenized** ▾ **Required:** Support on-demand payments by setting up Klarna's on-demand flow, enabling Partners to securely store payment details for recurring or variable usage scenarios, such as ride-sharing or other on-demand services (2.6.2.3.2 on-demand).

- **Physical Store** ▾ [Required](#): Implement Klarna for Partners with physical locations via your physical store solution, ensuring a unified experience across digital and physical channels. This includes compatibility with in-store devices, an optimized checkout flow for face-to-face transactions, and handling of instore returns of online purchases where applicable (2.6.3.3 Payments in Physical Store)
 - **Mobile** ▾ [Recommended](#): Use Klarna's Mobile SDK for native mobile app integrations, ensuring seamless in-app payment functionality that supports Klarna's full feature set while leveraging device-specific capabilities for improved performance and user experience (2.6.3.2 Payment request initiated with Klarna's MobileSDK).
- Ensure integration robustness by* integrating proactive communication across services and integration paths
- [Required](#): Set up webhooks to monitor payment statuses and retrieve confirmation tokens, ensuring timely updates and visibility into each transaction's lifecycle (2.4.2 Configure Klarna Webhooks).
- Deliver a post-purchase experience on par with direct Klarna integration:* Implement robust transaction management processes to support captures, updates, refunds, and voids.
- [Required](#): Follow Klarna's Transaction Management guidelines to handle all aspects of transaction management, including capturing payments, updating transaction data, issuing refunds, and voiding transactions as needed (2.8 Managing Payment Transactions).
 - [Required](#): Utilize webhooks for real-time notifications on transaction events, ensuring accurate transaction tracking and prompt action on changes (2.4.2 Configure Klarna Webhooks)
- Simplify dispute resolution:* Set up a dispute resolution process that enables partners to efficiently resolve payment disputes.
- **Dashboard** ▾ [Required](#): Use Klarna's Dispute API to manage dispute states and submit necessary documentation. (2.9 Dispute Handling).
 - **No Dashboard** ▾ [Required](#): Grant Partners access to Klarna's Partner Portal to allow handling of disputes directly. (2.7.1 Step 1: Grant access to Klarna's Partner Portal)
- Keep reconciliation easy:* Manage pricing configurations and reconcile Klarna settlements to ensure accurate payment handling.
- [Required](#): Configure and subscribe to settlement webhooks for timely event notifications (2.10.5.1 Settlement webhooks)
 - [Recommended](#): Integrate the Klarna Settlements API (2.10.4.1 Settlements API).
 - [Alternate flow](#): Set up Klarna SFTP (2.10.4.3 SFTP settlement reports)

2.1.5.4 Product Suite Interoperability

Enable interoperability across Klarna's product suite to ensure seamless integration across all features.

- Distribute Klarna Boost products:* Ensure all partners are able to access Klarna's Partner Portal to actively manage brand presentation and retrieve credentials.
- **Dashboard** ▾ [Required](#): Deep linking to Klarna Portal
 - **Dashboard** ▾ [Alternate](#): SAML access for your accounts to Klarna Portal. (2.7.1 Step 1: Grant Access to Klarna's Partner Portal).
 - **No Dashboard** ▾ [Alternate](#): Direct User Access to Klarna Portal via email provisioned in onboarding.



⚠ Only acceptable in cases where the Acquiring Partner is not able to support SAML login to Klarna Portal and does not have a Dashboard suited for deep linking.

- Enable Partners to benefit from conversion boost products to their end customers by supporting streamlined purchase flows.*
 - **Required:** Enable partners to pass `interoperability_token` in your “order creation” or “payment confirmation” endpoint directly with no earlier interaction via, enabling transaction confirmation. (2.7.2.2 Passing the Interoperability Token)
 - **Alternate:** if order is in Status=completed, present a UX which reflects this state, when the `interoperability_token` is passed in a “create” call. Allow the partner to explicitly call out the status in this interaction (2.6.1.2.3 Step 3: Read payment state and perform next action)

- Enable Partners to deliver a consistent and streamlined customer journey by retrieving and forwarding interoperability content to Klarna across all touchpoints.*
 - **Hosted** - **API Driven** - **Required:** Retrieve and pass the `interoperability_token` to Klarna in all interactions to ensure consistent session management and personalization for customers across platforms and devices (e.g., 2.7.2 Step 2: Consume Klarna Interoperability Token).
 - **Hosted** - **API Driven** - **Required:** Preselect Klarna where the Payment status indicates Klarna has previously authenticated a customer. (2.7.2 Step 2: Consume Klarna Interoperability Token)
 - **Hosted** - **Required:** Expose a way for a Partner to submit the `interoperability_token` and interoperability data to Klarna.
 - **Hosted** - **Required:** Use the `resolve()` function within Klarna’s SDK to set and manage the `interoperability_token` on the client side. This keeps the customer experience cohesive and reduces the need for customers to re-enter information.
 - **API Driven** - **Required:** Allow a Partner to submit the `interoperability_token` and interoperability data, passing this information to Klarna where provided by the Partner.
 - **Platforms & Plugins** - **Required:** Retrieve and pass the `interoperability_token` and data for platform plugins to Klarna (e.g., via 2.7.2.1.3 Case 3: Retrieving the Interoperability Token via Platform Plugins).

- Improve payment session outcomes and conversion: Enhance the accuracy and efficiency of payment sessions by ensuring that all relevant data is provided to Klarna for optimal processing.*
 - **Required:** Ensure your API accepts `klarna_interoperability_data` as a passthrough field to allow Partners to share key information related to the payment session. (2.7.3 Step 3: Consume Klarna Interoperability Data).
 - **Recommended:** Map any supplementary purchase information gathered by your APIs to the `supplementary_purchase_data` (SPD) object on payments to pass relevant data points to Klarna. (2.6.4.2 Supplementary Purchase Data).
 - **Recommended:** Educate Enterprise Partners to implement the Shopping Session API to share additional customer and transaction data points during the shopping journey.(2.7.5 Step 5: Educate Partners on Shopping Session API).

2.1.5.5 High Performance

Optimize performance by implementing rate limits, monitoring tools, and effective error handling for stability and efficiency.

- Gracefully handle volume spikes: support DDOS Protection and Rate Limiting to mitigate potential service disruptions due to flash sales or malicious behavior.*



- [Required](#): Handle Klarna rate limiting and throttling mechanisms to manage traffic spikes effectively (4.4 Rate Limiting).
 - [Recommended](#): Implement DDOS protection tools that can detect and mitigate high-frequency attacks. (4.3.2 DDOS Protection).
- Build a resilient integration to maintain stability, reliability, and high performance under varying loads*
- [Required](#): Use idempotency keys in POST and PATCH requests to prevent unintended duplication. (4.5.1 Idempotency).
 - [Required](#): Follow Klarna's error codes and types for consistent handling and quick recovery. (4.5.4 Error Handling).
 - [Recommended](#): Use a structured logging approach to track webhook verifications, storing signatures and relevant identifiers for troubleshooting and auditing purposes.
 - [Recommended](#): Perform regular load and performance testing of their integrations to ensure high availability and response times under peak traffic conditions
- Enable granular monitoring and streamlined troubleshooting*
- [Required](#): Set up real-time monitoring and alerting for quick issue detection and response. (4.5.3 Monitoring and Alerting).
 - [Recommended](#): Apply Partner tagging for each API operation to improve tracking and troubleshooting. (4.5.2 Tagging).
- Every Klarna customer has the same shopping experience across all integration approaches*
- [Required](#): Educate Partners on using interoperability features to maintain seamless customer experiences and pass Supplementary Purchase Data to Klarna.
 - [Required](#): Emphasize Klarna's conversion-boosting features such as On-site messaging, Express checkout, and Sign-in with Klarna, helping Partners understand how these features enhance customer interactions.
 - [Required](#): Guide Partners to include Klarna in key touchpoints (e.g., checkout pages, product pages), using dynamic presentation wherever possible to align with Klarna's brand guidelines.
 - [Required](#): Provide a clear overview of Klarna's value proposition, highlighting the benefits of increased conversions, repeat business, and customer reach. Use a pre-approved Klarna payment descriptor to prevent misinterpretation and maintain brand integrity.
- Partners are enabled to present Klarna in accordance with Klarna's best practices.*
- [Required](#): Document each step of the Klarna customer journey, from pre-purchase to post-purchase, allowing Partners to align their service experience with Klarna's best practices.
 - [Required](#): Include a full technical integration guide, covering API integration, platform plugins, mobile SDK, and in-store solutions. Ensure all methods for enabling Klarna are clearly defined for Partners of varying technical capabilities.
 - [Required](#): Provide access to Klarna branding assets, including logo lock-ups, badges, and messaging templates, allowing Partners to maintain brand consistency in their marketing and storefronts.
 - [Recommended](#): Use product matrices or interactive pages to clearly display geographical and product availability, ensuring Partners know where Klarna's services are supported.
- Public documentation is compliant with Klarna standards and aligned with agreed requirements, ensuring a consistent and high-quality experience for partners and users*



- Required: Complete a review with Klarna before releasing public documentation to ensure compliance with Klarna standards, clarity, and alignment.

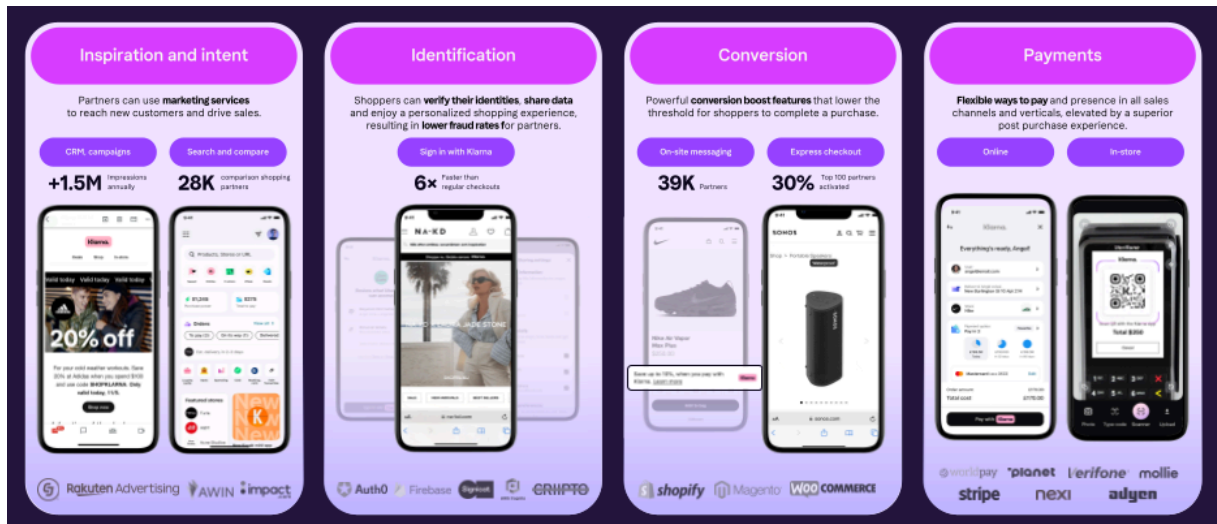
2.1.5.6 Payment Programs

- Enable Comprehensive and Flexible Payment Program Support:*
 - Required: Support and maintain all Klarna Payment Programs, ensuring consistent availability, flexibility, and dynamic updates across markets.
- Facilitate Secure and Tailored Payment Experiences:*
 - Required: Ensure secure, streamlined transactions by utilizing appropriate scopes and dynamically retrieved descriptors, aligned with the nature of the transaction and regulatory requirements.
- Ensure Global and Consistent Accessibility:*
 - Required: Payment Program offerings must remain globally accessible, providing consistency across markets without requiring technical changes to existing integrations when Klarna expands.
- Promote Accountability Through Dispute Management and Reporting:*
 - Required: Acquiring Partners and Partners are responsible for managing disputes and claims tied to Payment Programs, ensuring structured monitoring, prompt refunds, and accurate communications to resolve issues effectively.



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.

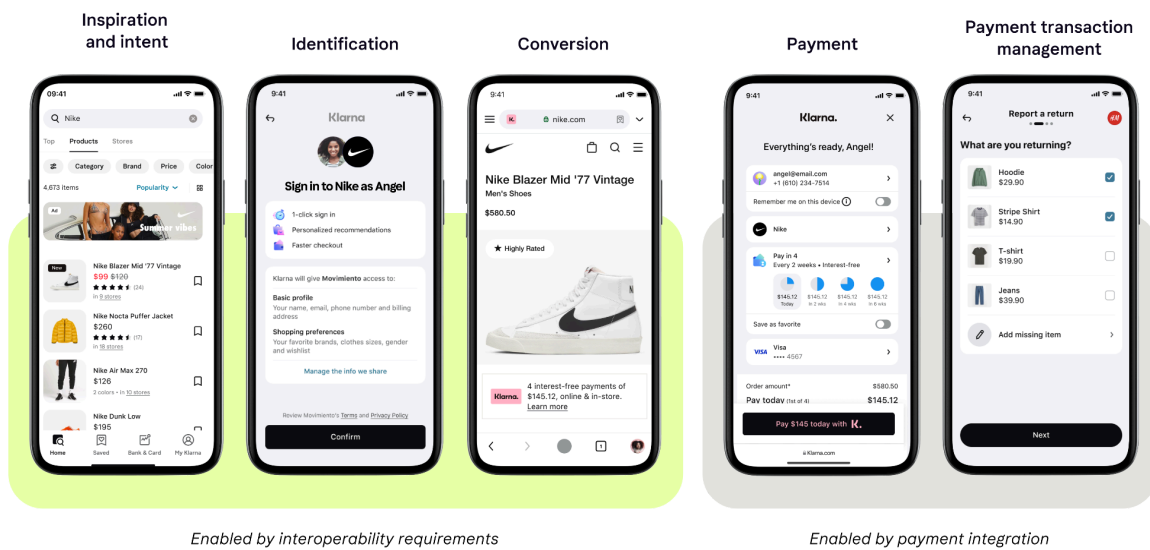
Partners have full access to the complete suite of Klarna products and services within this framework. By enabling the recommended integration flow outlined later in this document, Partners can achieve:

- **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

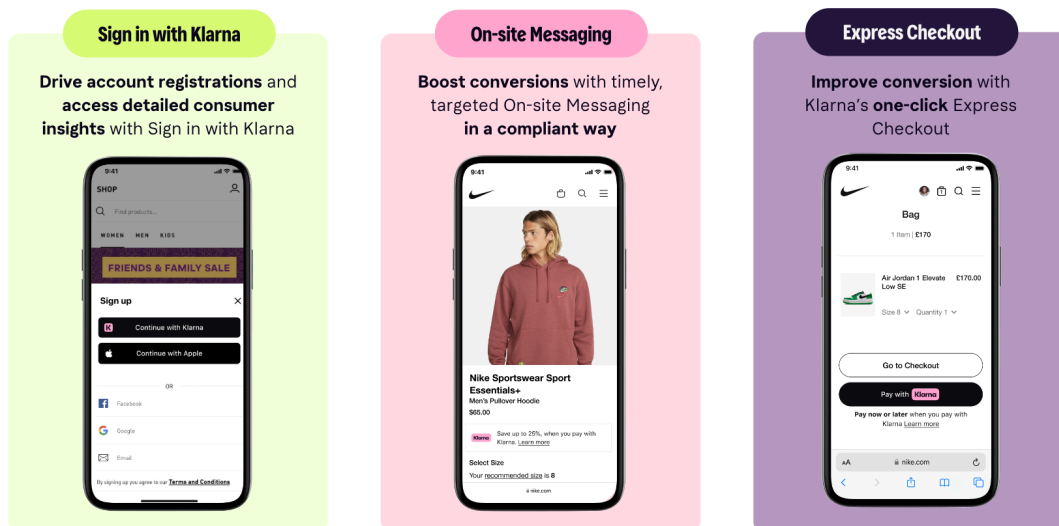


2.2.1.1 Accessing Klarna's Partner portal

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 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.

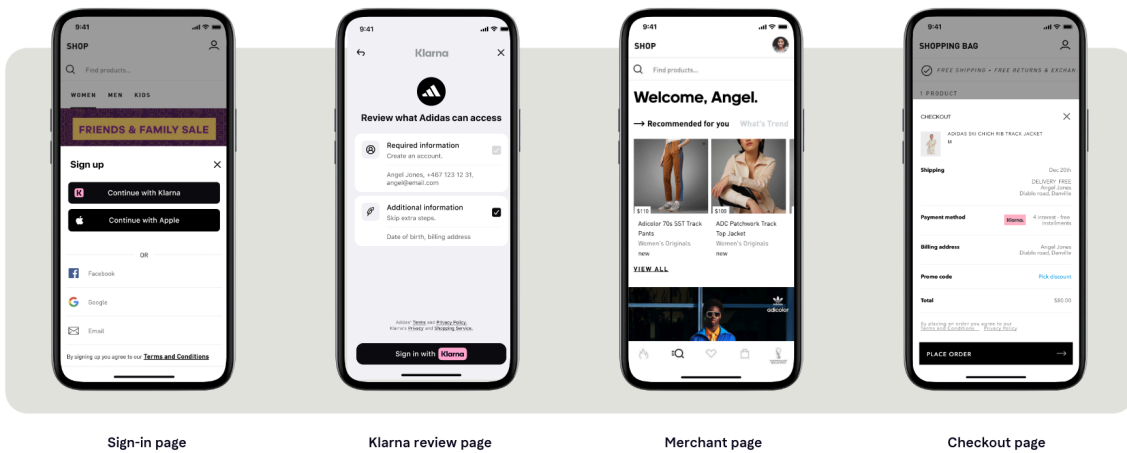


Tools to help Partners achieve maximum growth.

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 Track information. This allows Partners to identify their customers early in the shopping journey.



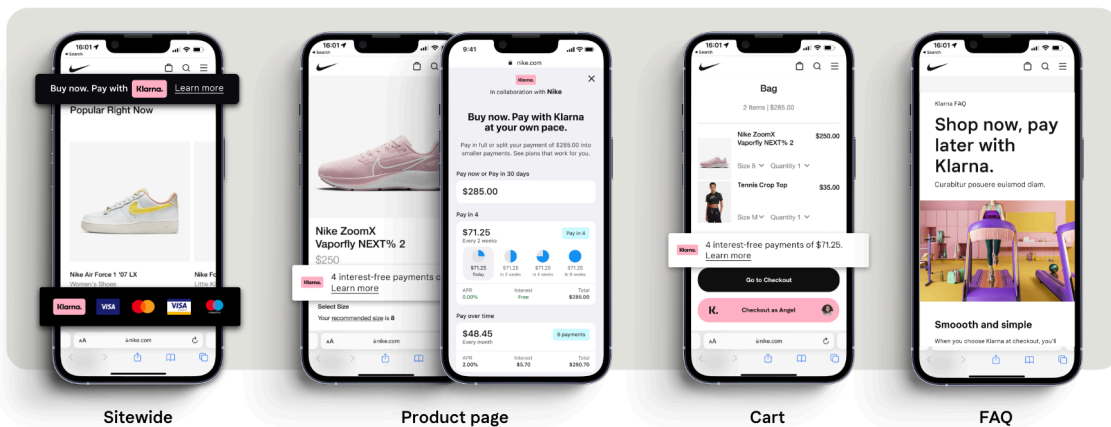
Sign-in page Klarna review page Merchant page Checkout page

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



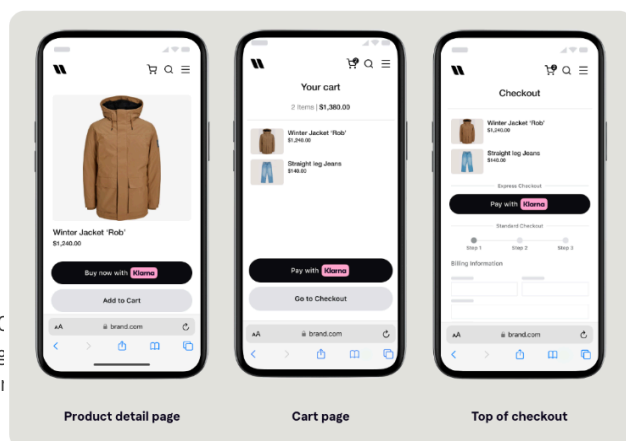
Sitewide Product page Cart FAQ

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 customers 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 and SDKs enable you to dynamically display accurate payment descriptors based on your account settings and transaction specifics. These tools provide crucial information and visuals, ensuring global compliance and helping you effectively showcase Klarna-branded elements to enhance customer conversion rates.

Technical details on confirming Klarna availability is available in [Display Klarna](#).

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 four main items to consider:

1 Payment descriptor

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

2 Payment subheader

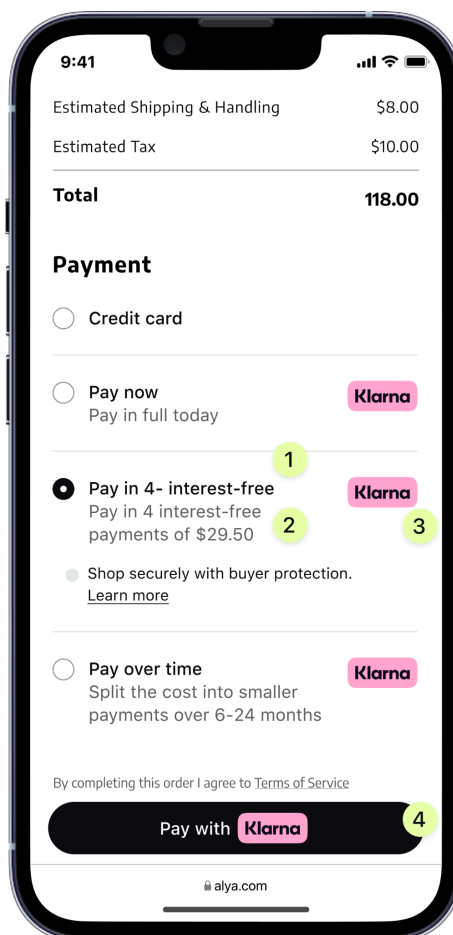
More detailed breakdown of the value of the option presented. Provided in the response.

3 Klarna badge

Klarna logo. Provided in the response.

4 Pay with Klarna button

Klarna Payment Button, replacing the default when Klarna is selected.



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.

2.3.2.3 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, 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.

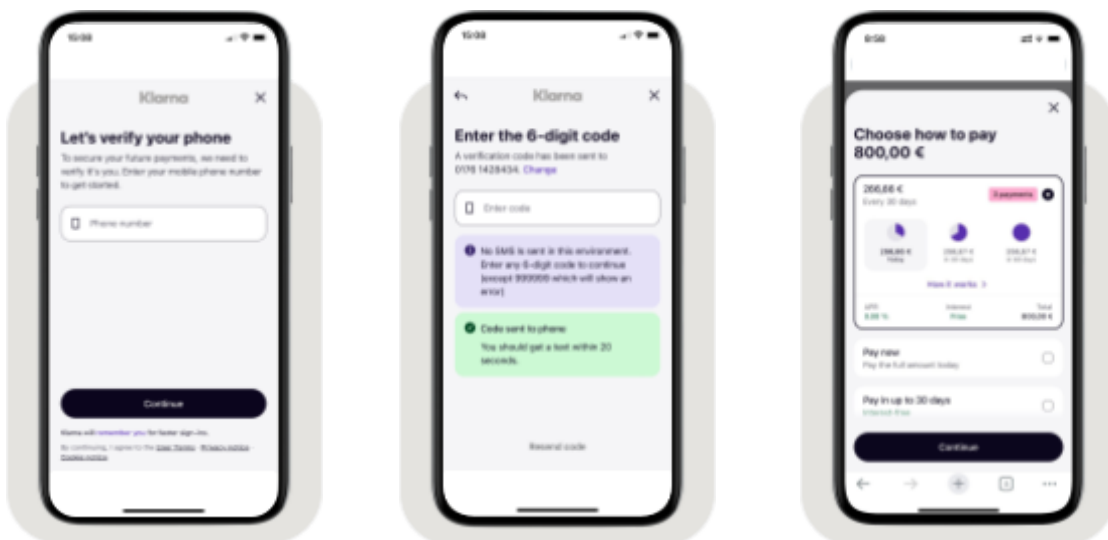


2.3.3 Payment flow

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:

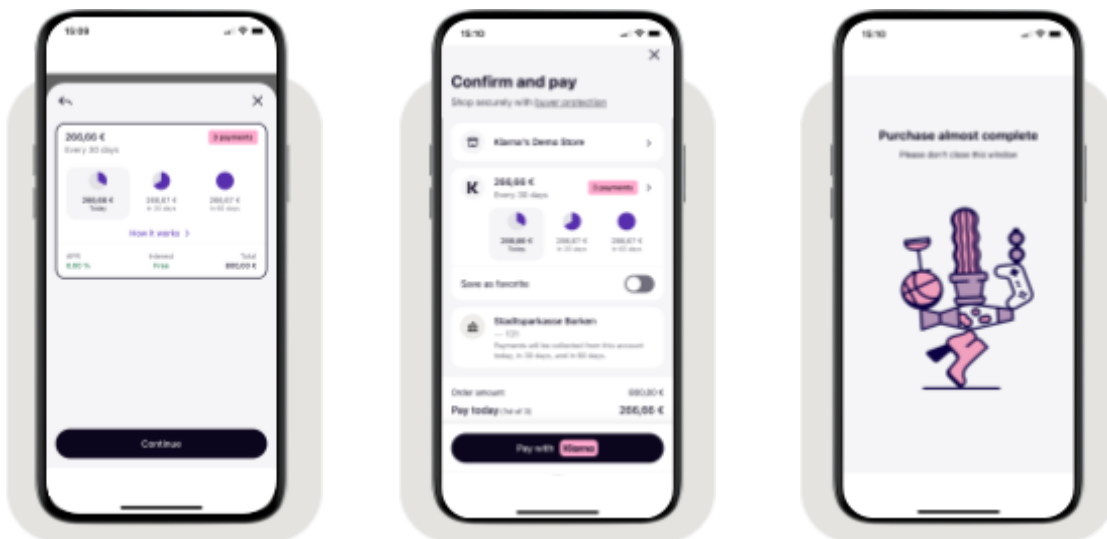
- [Client-side integration](#)
- [Server-side integration](#)

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 [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

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:

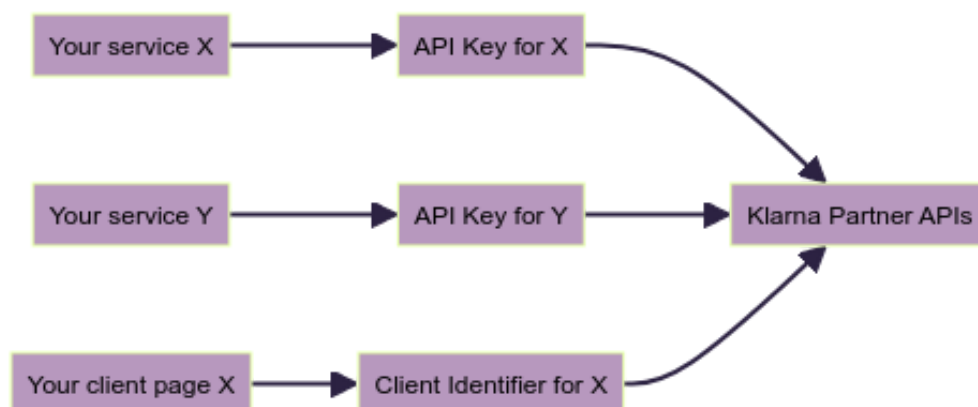
- **API keys:** Are used to authenticate server-side REST API requests towards Klarna. In addition, Klarna may use them to identify the source account.
 - Structure: `klarna_<live|test>_<api>_<random>`
- **Client IDs:** Are used to authenticate client-side interactions towards Klarna's SDK.
 - Structure: `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 [Security](#) 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 the event of a security breach, **assign unique credentials to each service** and use the `description` field to clearly define each credential's purpose. This approach simplifies credential management and ensures that if one credential is compromised or needs to be deactivated, it won't affect the others, maintaining 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 </v2/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 /v2/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 [Getting started with Klarna 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": "v2",
    "occurred_at": "2024-01-01T12:00:00Z",
    "subject_account_id": "{{account-specific ID}}",
    "recipient_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

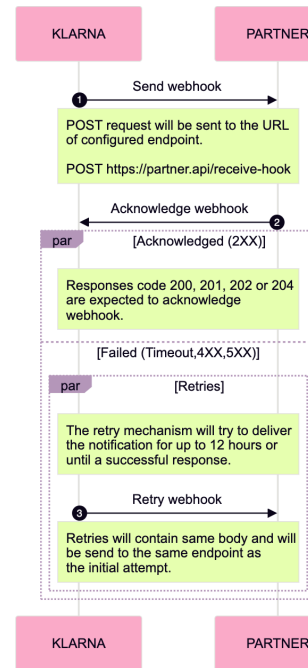
If no successful response is received after the final retry attempt at 12 hours, the notification will be considered permanently failed.

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.

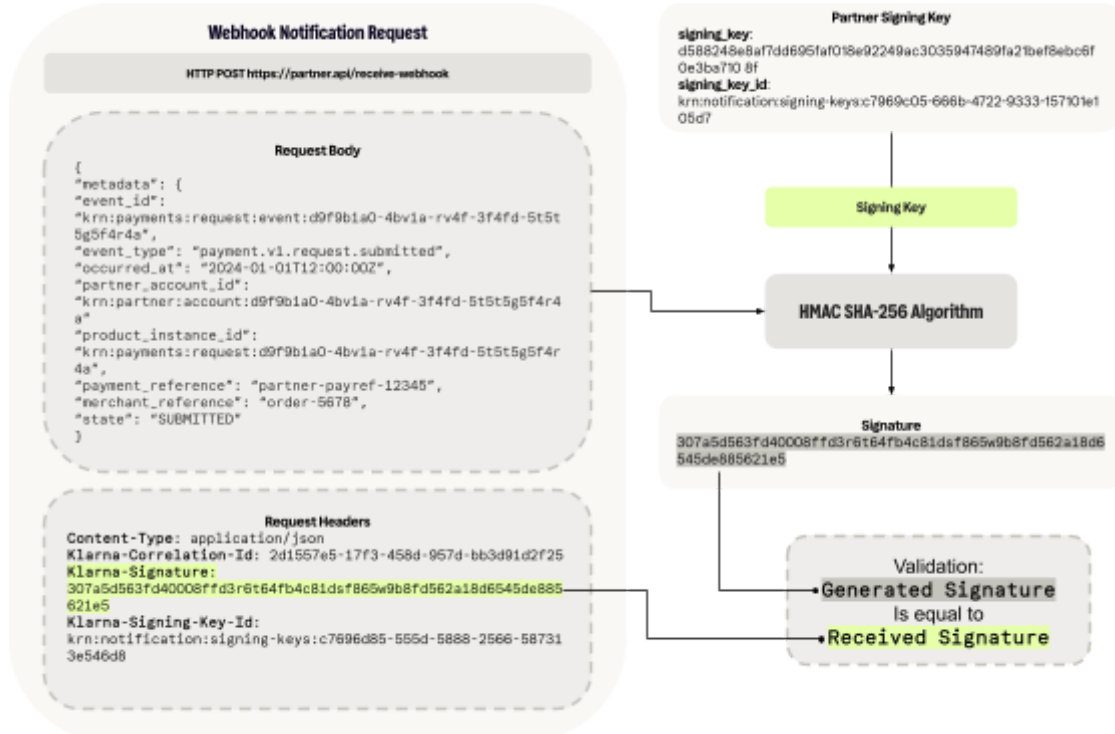
⚠ Important: After generating signing keys, make sure to securely store them immediately. You will not be able to retrieve these keys again for webhook validation purposes.

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 /v2/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.confirmed
- payment.request.state-change.canceled

- `payment.request.state-change.expired`
- `payment.transaction.state-change.authorized`
- `payment.transaction.state-change.completed`
- `payment.transaction.state-change.expired`
- `payment.transaction.state-change.closed`
- `payment.transaction.captured`
- `payment.transaction.refunded`
- `payment.transaction.chargeback`
- `payment.dispute.state-change.pre-arbitration`
- `payment.dispute.state-change.arbitration-pending`
- `payment.dispute.state-change.merchant-evidence-pending`
- `payment.dispute.state-change.closed`
- `payment.dispute.updated.disputed-amount-updated`
- `Payment.dispute.updated.arbitration-deadline-extended`

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](#)
- [Payment Transaction webhooks](#)
- [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.confirmed",
    "event_id": "d9f9b1a0-5b1a-4b0e-9b0a-9e9b1a0d5b1a",
    "event_version": "v2",
    "occurred_at": "2024-01-01T12:00:00Z",
    "recipient_account_id": "krn:partner:global:account:live:LPW1903",
    "subject_account_id": "krn:partner:global:account:live:LPW1903",
    "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_request_reference": "partner-payref-1234",
    "state": "CONFIRMED",
    "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

- /v2/accounts/{account_id}/extra-account-data/settlement-information
- </v2/distribution/products/payment/price-plans>

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

Release notes

 Further configuration points may become available with future releases.

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.

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

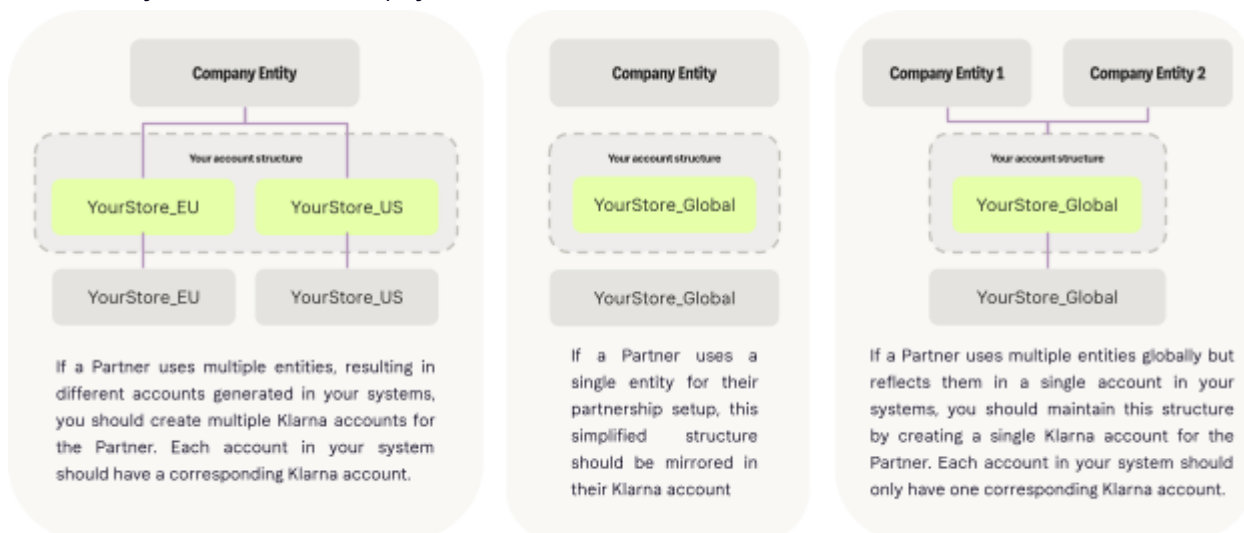
2.5.1 Step 1: Determine account structure

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 be able to enable Klarna without additional effort 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 will be added in future releases

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:

2.5.1.1.1 Retail

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



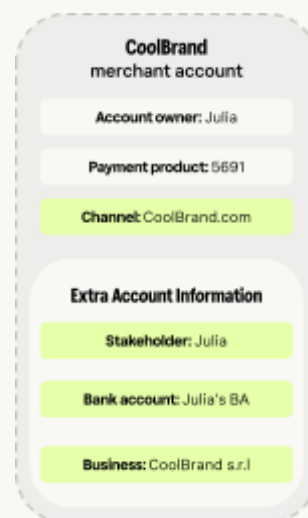
Example 1:

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

 17 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.

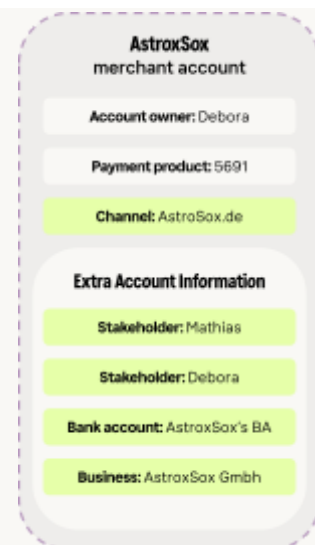
If you have the list of MCCs the partner is using at the point of onboarding, those MCCs can be passed on the *enforced MCCs* list as part of the payment product. When the enforced MCC list is present, providing an MCC that is not the list when initiating a transaction may result in the transaction being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



JavaScript

```
{
  "account_reference": "M12378999999",
  "account_name": "AstroxSox",
  "account_owner": {
    "given_name": "Mathias",
    "family_name": "Doe",
    "email": "Mathias@AstroxSox.de",
    "phone": "+49555555555"
  },
}
```



```

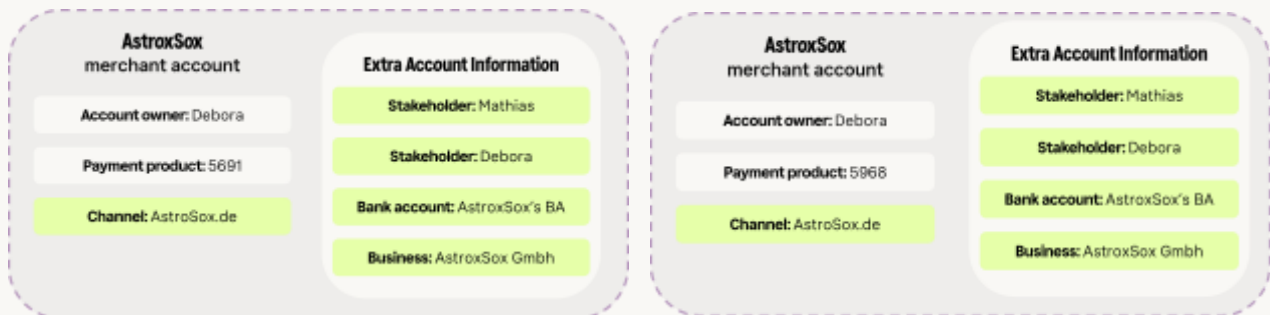
    "products": [
      {
        "distribution_profile_id":
        "krn:partner:global:account:distribution-profile:206bbb83-9b6e-46fa-940d-337153c04a5
        8",
        "type": "PAYMENT",
        "default_merchant_category_code": "5691",
        "enforced_merchant_category_codes": ["5691", "5641"]
      }
    ],
    "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": [

```

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
      -profile:206bbb83-9b6e-46fa-940d-337153c
      04a58",
      "type": "PAYMENT",
      "merchant_category_code": "5691"
    },
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://AstroSox.de"
        ]
      }
    ]
  }
}

```

```

    {
      "distribution_profile_id":
      "krn:partner:global:account:distribution
      -profile:206bbb83-9b6e-46fa-940d-337153c
      04a58",
      "type": "PAYMENT",
      "merchant_category_code": "5641"
    },
  ],
  "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.

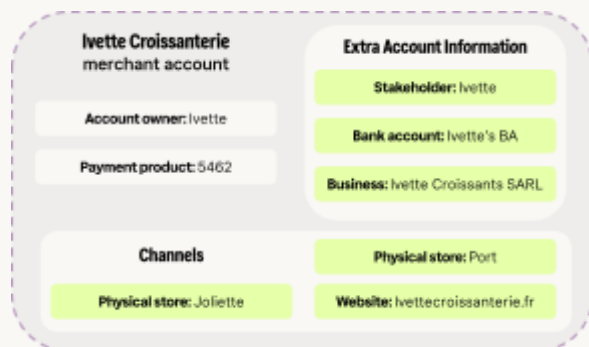
Example 3

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": "+4955555555"
  },
  "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": "+49555555555"
  },
  "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",
      }
    ]
  }
}
```

2nd:

```
JavaScript
{
  "account_reference": "M123789922223",
  "account_name": "Liam WTB",
  "account_owner": {
    "given_name": "Liam",
    "family_name": "Doe",
    "email": "liam.franchisee@wtb.se",
    "phone": "+49555555555"
  },
  "products": [
    {
      "distribution_profile_id":
      "krn:partner:global:account:distribution-
      profile:206bbb83-9b6e-46fa-940d-337153c04
      a58",
      "type": "PAYMENT",
      "merchant_category_code": "5812"
    }
  ],
  "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": "+49555555555"
  },
  "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. 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 are 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. Providing incomplete data increases the perceived risk associated with your Partners.

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.

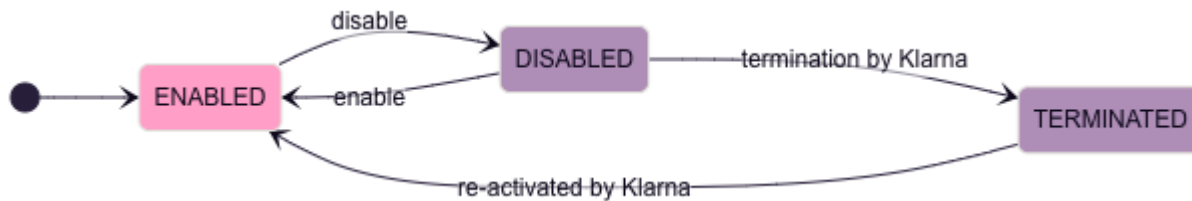
2.5.3 Step 3: Manage Partner payment products

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. Klarna requires you to provide the most up-to-date data for each account as soon as it is available to you.

For the most robust capabilities and safer handling of these resources, we recommend you use resources-specific endpoints when making updates.

Alternate Flow
Klarna provides a GET all subresource and PATCH all subresource endpoints for handling of all configurations within an account in a single call. This approach introduces more latency and complicates error handling (as if one resource update fails all must be re-updated) and has some limitations as outlined within the API specs, and is therefore not the recommended flow.

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:

- Suspend: [/v2/accounts/{account_id}/products/payment/{product_id}/disable](#)
- Enable: [/v2/accounts/{account_id}/products/payment/{product_id}/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 webhooks](#) 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

17 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.

2.5.3.2.2 Re-enabling a product

In case a disabled payment needs to be reenabled, 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 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 added to it. All products within an account are fully disabled as a part of this status.

It is essential to configure [account lifecycle webhooks](#) to receive notifications about status changes in Partner accounts and Payment Products. 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": "v2",
"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

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.

Hosted Integration

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.mjs, 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.

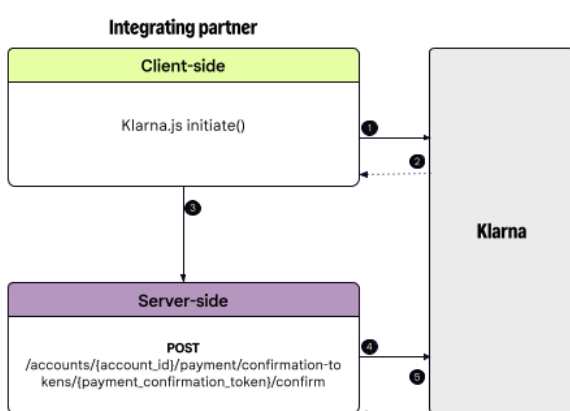
Blended Integration

For offerings where the Acquiring Partner integrates both client-side and server-side elements of Klarna's services, a blended integration model is recommended. This involves leveraging Klarna's Web SDK for client-side functionalities, such as Express Checkout or Sign-in with Klarna, while utilizing the Partner Product API for backend operations where appropriate according to architectural limitations.

Alternate server-side only integration

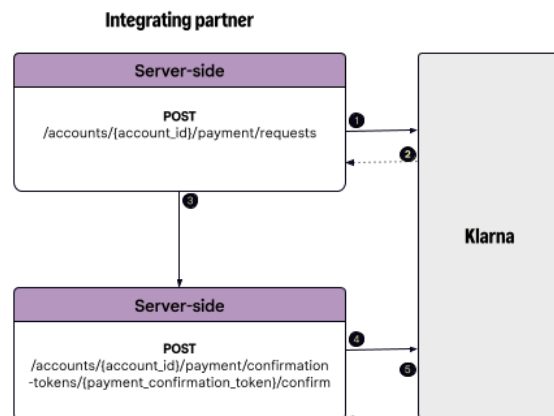
For offerings that solely provide server-side integration to Partners, where the Partner owns the client-side integration of Klarna, you should adopt the Partner product API and expose redirect links to your Partners. This ensures compliance while allowing for flexibility to accommodate your integration 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.mjs `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.

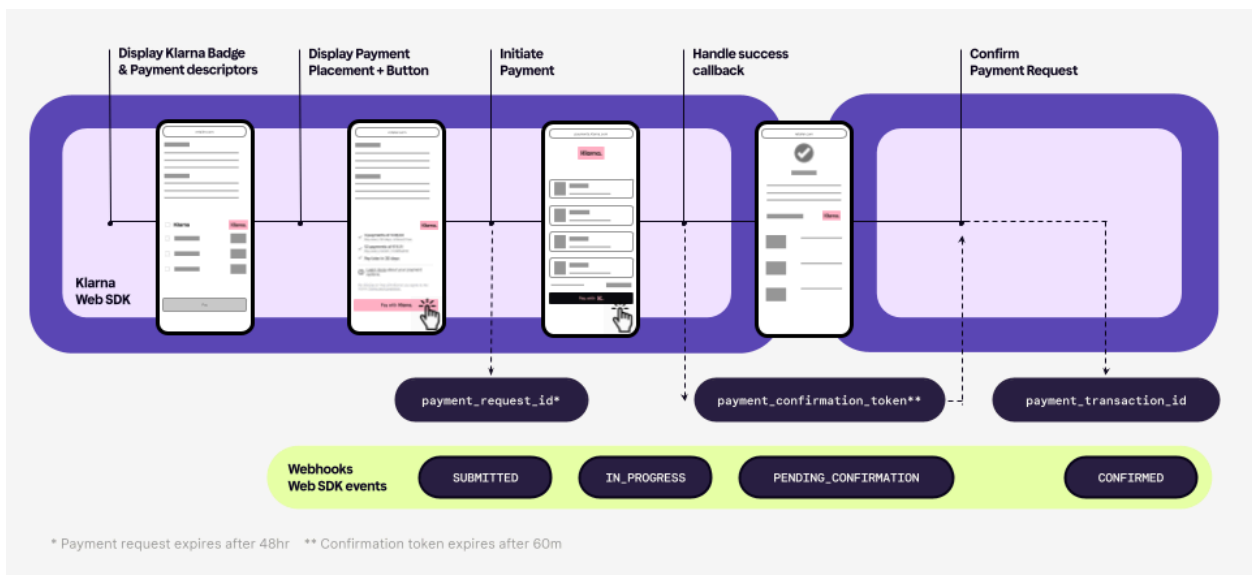


Using Klarna's `interoperability_token` to create a unified and seamless shopping experience is required, especially when your integration includes both client-side and server-side elements. This identifier is provided by the Klarna web SDK in response to any initiating action where an existing `interoperability_token` is not provided. By providing this parameter to the Partner (when you manage the initiation of the web SDK) and enabling the Partner to include it in all requests, you integrate these elements, enhancing personalization, security, and cohesion in the checkout process. This reduces friction, leading to higher customer satisfaction and increased conversion rates. For more information on using Klarna's `interoperability_token` to improve the customer experience, see the [Enable Interoperability](#) section.

2.6.1 Online store transaction

A client-side integration via the Javascript library `Klarna.mjs` 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:



2.6.1.1 Step 1: Include the Klarna SDK

To ensure optimal performance and security, include the `Klarna.mjs` script on every page that displays a Klarna component.

⚠️ Always load the script directly from <https://js.klarna.com/web-sdk/v2/klarna.mjs>; 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's SDK 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/v2/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.

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

To integrate Klarna Payment Services seamlessly it is critical as the first step to confirm the availability of Klarna Payment Services and customize the payment messaging by using the `resolve()` and `presentment()` methods.

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

Invoke the `resolve()` method to find out if a payment is available for a specific combination of customer country, currency, and amount. Relying on this method reduces the effort required for you to roll out to new markets as they become available, 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. Klarna must be selected as a primary payment method if the state is `PAYMENT_PREPARED` or `PAYMENT_PENDING_CONFIRMATION`.

JavaScript

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



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

For technical information on how to get content regarding the presentation of Klarna in checkout, refer to [Payment presentment instruction](#). Information on Klarna's branding expectations are available in [How to present Klarna in checkout](#).

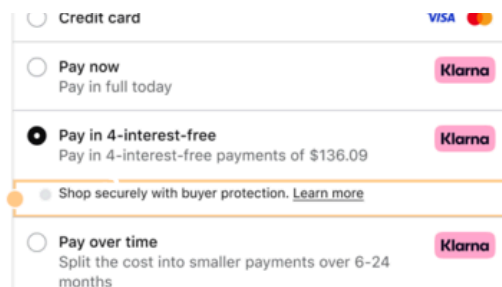
Consult the [SDK reference](#) for a complete description of the request body parameters and [Tokenized Payments](#) for details on how to handle subscription and on-demand purchases.

2.6.1.1.2.3 - Display the Klarna checkout placement

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

Checkout Placement

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

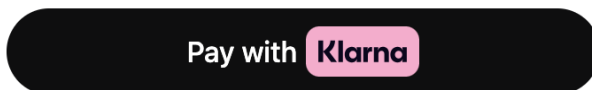


JavaScript

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

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

Klarna requires the Klarna payment button be used to initiate the payment request 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 changing the default payment button to Klarna's payment button when a customer selects Klarna, and matching the appearance of the button to the Partner page using themes.

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

Unset

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



2.6.1.1.3.2 Use the klarna.mjs 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');
```

2.6.1.1.3.3 Prepare the button click event configuration

Configure the InteractionMode

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

interactionMode	Definition
DEVICE_BEST	This is the default value and recommended . Klarna automatically selects the best flow depending on the device: <ul style="list-style-type: none">• Mobile: REDIRECT• Desktop: Popup window if possible, fallback to REDIRECT.• Native webview (mobile app) - REDIRECT Note: <code>config.redirectUrl</code> is required in the payment request for this interaction mode
ON_PAGE	This value should be used only if redirection is not possible. The transaction happens on the same page by using a modal window if possible, if not then it will fallback to fullscreen iframe.
REDIRECT	Only redirect flow. Note: <code>config.redirectUrl</code> is required in the payment request for this interaction mode

Generate a Redirect URL

The `redirectUrl` guides the customer back to your online store after a completed or canceled payment request. For mobile integrations, ensure the `redirectUrl` is a deep link. For more information on Mobile implementations, refer to [Payments in mobile apps](#).

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.payment_confirmation_token}&request_id={klarna.payment_request.id}&state={klarna.payment_request.state}&reference={klarna.payment_request.payment_request_reference}"
}

```

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_request_reference}	The provided reference to the payment request.	partner-payment-reference-12345

Payment request updates

A payment request can be initiated either directly by the Klarna Partner, 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, it may be allowed to persist if an existing `payment_request_id` is passed.
- 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? | string, options): PaymentRequest`
- `initiate(paymentRequestData?, options?): Promise<void>`

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

Parameter	Definition
<code>paymentAmount</code>	Total payment amount

Parameter	Definition
currency	3-letter ISO 4217 currency code
paymentRequestReference (optional)	Reference to the payment session or equivalent resource created on your side. This will be exposed in payment request webhooks (payment.request.*) for the purpose of correlating your resource with the Klarna Payment Request.
config (optional)	Configuration object for the payment request.
config.redirectUrl (optional)	Payment flow redirection URL. The customer will always be redirected to this URL after the purchase is approved, if running in redirection mode the customer will also be redirected to this URL on a failed authorization.
config.paymentOptionId	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. 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.
supplementaryPurchaseData (optional)	An object encapsulating additional information related to a purchase transaction, providing comprehensive details to support and enhance the core purchase data.
supplementaryPurchaseData.purchaseReference (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.
supplementaryPurchaseData.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
supplementaryPurchaseData.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.
supplementaryPurchaseData.lineItems (optional)	Detailed line item information of the purchase. This data is used for fraud detection and improving the customer experience in the Klarna app. The total sum of the line items must match the payment amount. Line items stand at the crossroads of Klarna's operations, improving our fraud detection and underwriting capabilities, enhancing the consumer's post-purchase experience through streamlined disputes, returns and more. By dissecting transactions into individual units of purchase, we provide a detailed map of the consumer's buying behavior, facilitating a more personalized and efficient service in the Klarna app.

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 in the [Klarna SDK reference](#).

When the customer enters Klarna's [payment flow](#), they can select their payment method and provide any required information. This 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 not be subject to the authentication step in the funnel.

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

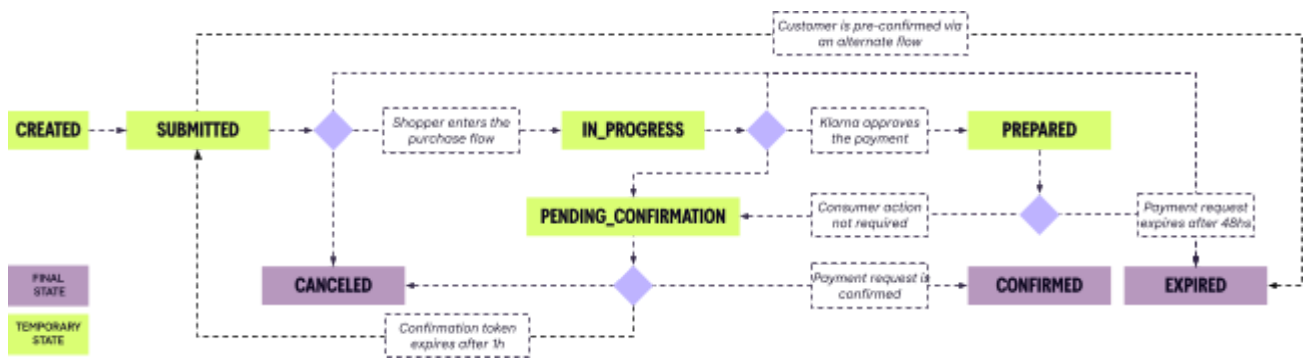
During the checkout process, the payment request will transition to various states. Find below an overview of the possible transaction states together with a transition diagram:

Payment request state

State	Definition	Next action
CREATED	Payment request has been created client-side, can freely be modified (Not valid for server side integration)	Submit or initiate the payment request by calling the respective method
SUBMITTED	Payment request has been submitted to Klarna's backend and ready to be initiated, request can be modified but must be synchronized	Redirect to the payment distribution URL using <code>state_context.payment_distribution.url</code>
IN_PROGRESS	The payment flow is in progress (customer is inside the purchase flow)	Listen to webhooks for state changes.
PENDING_CONFIRMATION	The payment flow has successfully been completed by the customer and is pending final confirmation to complete the payment request.	Confirm the payment request using the <code>state_context.payment_confirmation_token</code>
EXPIRED	The payment request has expired (t≥48h). This is a final state.	No action.
CONFIRMED	The payment request is confirmed - and a payment transaction has been created. This is a final state.	Read and store information inside the <code>state_context</code>
CANCELED	The payment request has been canceled by the integrator. The <code>payment_confirmation_token</code> can only be CANCELED until the request is CONFIRMED. After confirmation, the cancellation is no longer possible. This is a final state.	No action.

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.

2.6.1.1.4.1 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": "v2",
    "occurred_at": "2024-05-27T14:41:30Z",
    "subject_account_id":
"krn:partner:global:account:test:d6312901-1056-4231-8adb-d5abea7f3f8c",
    "recipient_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_request_reference": "09e7a0f1-4207-4abe-b472-64559b14cc80",
    "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.

2.6.1.1.4.2 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)
})

```

2.6.1.1.4.3 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 /v2/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",
  "expires_at": "2024-01-02T13:00:00Z",
  "created_at": "2024-01-01T12:00:00Z",
  "updated_at": "2024-01-01T13:00:00Z"
}
```

2.6.1.1.4.4 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_request_referenc
  e}"
}
```

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: CONFIRMED
- Payment request 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=CONFIRMED&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. Consult the [API reference](#) for a complete description of the request body parameters.

2.6.1.1.5 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:
[/v2/accounts/{account_id}/payment/token/charge](#).

Mandatory Request Parameter	Definition
<code>currency</code>	The currency in which the transaction is made.
<code>payment_amount</code>	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:

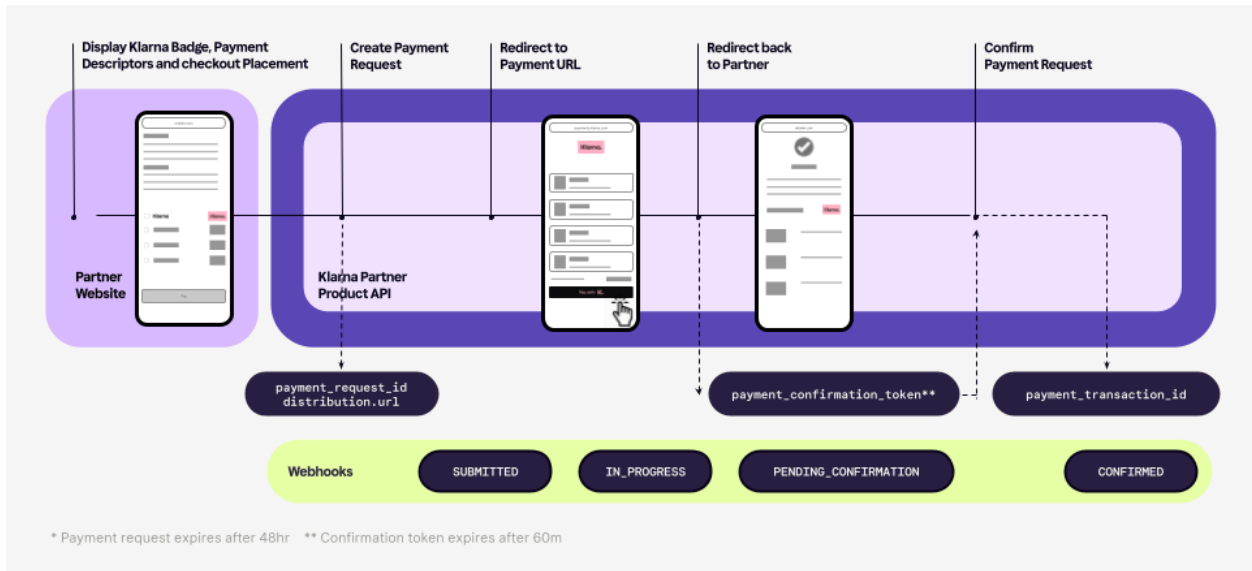
```
Unset
{
  "state_context": {
    "payment_transaction_id":
"krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
    "payment_pricing": {
      "applicable_rate": {
        "fixed": 0,
        "variable": 1700000,
        "unit": "CAPTURE"
      }
    },
    "previous_state": "PENDING_CONFIRMATION",
    "payment_request_id": "krn:payment:eu1:request:bebeabea-5651-67a4-a843-106cc3c9616a",
    "state": "CONFIRMED",
    "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:



2.6.1.2.1 Step 1: Check Klarna payment interoperability status 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.mjs 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 presentment() method of klarna.mjs 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 the response.

2 Payment subheader

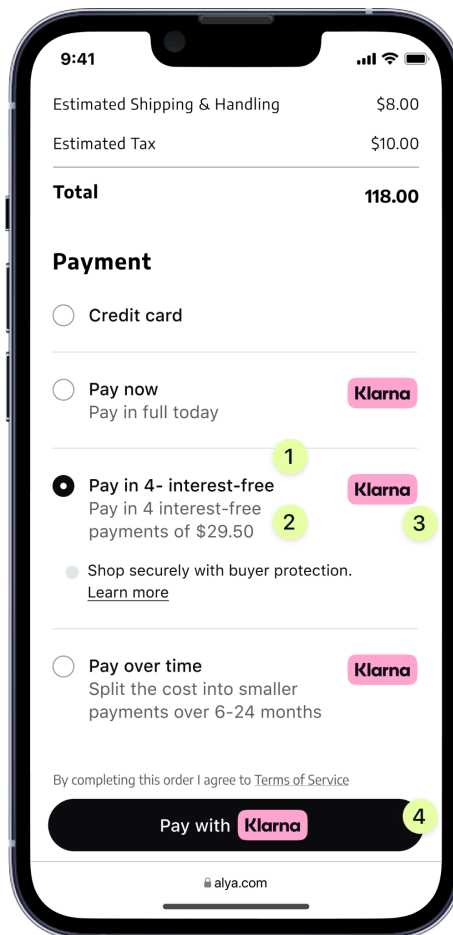
More detailed breakdown of the value of the option presented. Provided in the response.

3 Klarna badge

Klarna logo. Provided in the response.

4 Pay with Klarna button

Klarna Payment Button, replacing the default when Klarna is selected.



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 `presentment` 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 presented as a single payment option (using `category_preference=KLARNA`). 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
<code>subscription_interval (optional)</code>	Indicates the cadence of the subscription if <code>message_preference=SUBSCRIPTION</code> , 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=KLARNA` is specified, Klarna will return the messaging copy for the **Single-option approach**


Find more information about available approaches [here](#).



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

The payment_option_id is a parameter that should be used when initiating the payment request, as it allows you, as the acquiring partner, to preselect the corresponding payment option when the customer enters the [purchase flow](#).

Release notes

 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",
            "value": ""
          }
        ]
      }
    }
  ]
}
```

```

    "url": "...",
    "label": "Klarna logo"
  }
]
},
...
]}

```

Use the Partner Product Descriptor API

Release notes

 This request will be available in future releases.

Use the [GET /v2/accounts/{account_id}/payment/messaging/payment-descriptors](#) to retrieve relevant payment descriptors for displaying Klarna in your checkout.

Query Parameter	Definition
locale	Locale to use for returned content. BCP 47 (concatenation of language code (ISO 639-1 format) + "-" + country code (ISO 3166-1 alpha-2 format)) Example: en-US
payment_amount	Total amount of a one-off purchase, including tax and any available discounts. The value should be in non-negative minor units. Eg: 25 Dollars should be 2500.
category_preference (optional)	Indicates the preferred payment option. Enum: <ul style="list-style-type: none"> • PAY_NOW • PAY_LATER • PAY_OVER_TIME • KLARNA

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.
- **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": "czxsa4324132dsaddscxzzxs",
      "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"
          }
        ]
      }
    }
  ]
}
```

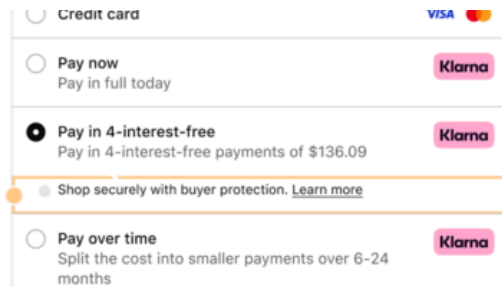
2.6.1.2.1.3 Display the Klarna checkout placement

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 /v2/accounts/{account_id}/payment/messaging/checkout, the response will contain the necessary element for Partners to create the checkout placement themselves.

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 /v2/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
interoperability.interoperability_token	Unique identifier for a shopping session.	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzaG9wcGluZ19zZXNzaW9uX2kljoia3JuaW50b3BwaW5nOmV1MTpzZ...
config.payment_option_id	Specifies a payment option to be pre-selected in the purchase flow.	czxsa4324132dsaddsxzxzs
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": "USD",
  "payment_amount": 1000,
  "interoperability": {
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzaG9wcGluZ19zZXNzaW9uX2kljoia3JuaW50b3BwaW5nOmV1MTpzZ..."
  },
  "config": {
    "payment_option_id": "czxsa4324132dsaddsxzxzs",
    "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": {
    "payment_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: Read payment state and perform next action

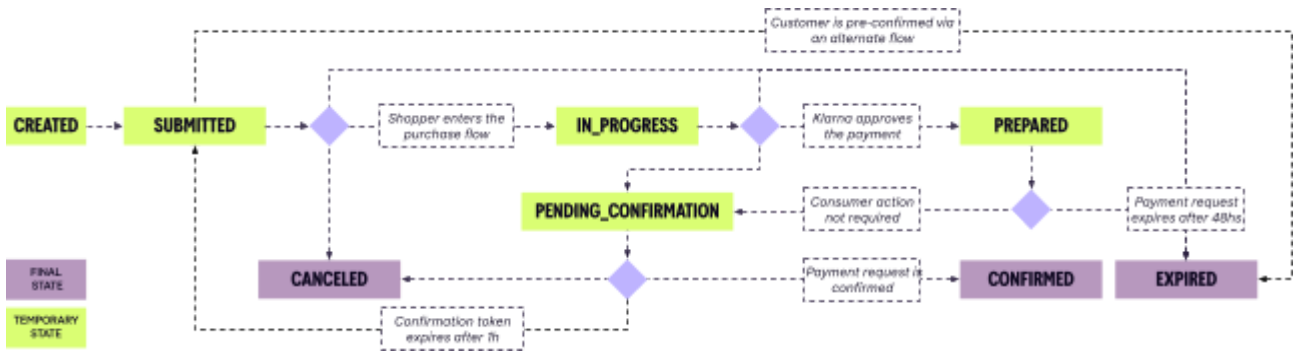
This section gives an overview of possible payment request states for a server-side initiated payment request. Depending on the state, the Partner should decide what action to perform next. After creating a payment request, it can end up at either SUBMITTED or AUTHORIZED state.

Payment request state definitions

State	Definition	Next action
SUBMITTED	Payment request has been submitted to Klarna's backend and ready to be initiated, request can be modified but must be synchronized	Redirect to the payment distribution URL using <code>state_context.payment_distribution.url</code>
IN_PROGRESS	The payment flow is in progress (customer is inside the purchase flow)	Listen to webhooks for state changes.
PENDING_CONFIRMATION	The payment flow has successfully been completed by the customer and is pending final confirmation to complete the payment request.	Confirm the payment request using the <code>state_context.payment_confirmation_token</code>
EXPIRED	The payment request has expired (t≥48h). This is a final state.	No action.
CONFIRMED	The payment request is confirmed - and a payment transaction has been created. This is a final state.	Read and store information inside the <code>state_context</code>
CANCELED	The payment request has been canceled by the integrator. The <code>payment_confirmation_token</code> can only be CANCELED until the request is CONFIRMED. After confirmed, the cancellation is no longer possible. This is a final state.	No action.



Payment request state diagram



2.6.1.2.4 Step 4: Redirect the customer to Klarna purchase flow

If the payment request ends up at **SUBMITTED**, the Partner should redirect the customer to the start link provided in the response. This link is located at `state_context.payment_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}"
}
```

2.6.1.2.5 Step 5: 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.

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.

2.6.1.2.4.1 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 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": "v2",
    "occurred_at": "2024-01-01T12:00:00Z",
    "subject_account_id":
      "krn:partner:global:account:test:d6312901-1056-4231-8adb-d5abea7f3f8c",
    "recipient_account_id":
      "krn:partner:global:account:test:d6312901-1056-4231-8adb-d5abea7f3f8c",
    "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_request_reference": "partner-payref-1234",
    "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 [/v2/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",

```

2.6.1.2.4.3 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
<code>{klarna.payment_request.payment_confirmation_token}</code>	The confirmation token, available when a transaction is successfully completed, and used to confirm the transaction.	<code>krr:payment:eu1:confirmation-token:51bbf3db-940e-5cb3-9003-381f0cd731b7</code>
<code>{klarna.payment_request.id}</code>	Klarna Payment Request identifier. Used in management of a Payment Request.	<code>bebeabea-5651-67a4-a843-106cc3c9616a</code>
<code>{klarna.payment_request.state}</code>	State of the payment request - may be used as a hint.	<code>CONFIRMED</code>
<code>{klarna.payment_request.payment_request_reference}</code>	The provided reference to the payment request.	<code>partner-payment-reference-12345</code>
<code>{klarna.payment_request.referrer}</code>	URL where the authorization started.	<code>https://partner.example/pdp</code>

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: CONFIRMED
- 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=CONFIRMED&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.

2.6.1.2.4.4 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 [/v2/accounts/{account_id}/payment/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"
}
```

2.6.1.2.6 Step 6: 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:

[/v2/accounts/{account_id}/payment/confirmation-tokens/{payment_confirmation_token}/confirm](#).

Mandatory Request Parameter	Definition
<code>currency</code>	The currency in which the transaction is made.
<code>payment_amount</code>	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"
  },
  "previous_state": "PENDING_CONFIRMATION",
  "payment_request_id": "krn:payment:eu1:request:bebeabea-5651-67a4-a843-106cc3c9616a",
  "state": "CONFIRMED",
  "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.• 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>	The "Customer login" scope can be used to retrieve customers' personal data from Klarna. Retrievable data includes: <ul style="list-style-type: none">• customer contact details,• transaction information• Shopping Profile data.
<code>payment:customer_present</code>	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. 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.
<code>payment:customer_not_present</code>	Tokens with the Customer Not Present on Payment scope can be used to complete transactions on behalf of the customer, initiated by the Partner. 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.

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.mjs 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.

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.

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
<code>currency</code>	The currency in which the transaction is made.
<code>payment_amount</code>	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.
<code>subscription.subscription_reference</code>	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.
<code>subscription.name</code>	The name of the subscription. This will be shown to the customer in the payment flow and in the Klarna App.
<code>subscription.subscription_plan</code>	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.

Parameter	Definition
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 state_context.customer_token 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

 17 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_request_reference": "partner-payref-1234",
  "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/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 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_request_reference": "partner-payref-1234",
  "config": {
    "request_customer_token": {
      "scopes": [ 'payment:customer_not_present' ]
    },
    "redirect_url": "https://partner.example/redirect?id={klarna.payment_request.id}",
  }
}
```

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 [create a 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": "CONFIRMED",
  "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.

Aside from adding `message_preference`, presenting Klarna during checkout remains consistent with the standard process described in [Online store transactions](#). Where possible, including the estimated `purchase_interval` is recommended to provide more context for Klarna's risk engines.

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>	The total amount to be charged, matching the sum of the <code>line_items[].total_line_amount</code> . 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.
<code>ondemand_profile.reference</code>	Reference to this on-demand profile, the same reference must be passed in when charging for this on-demand service. 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.
<code>ondemand_profile.details</code>	Specifies the expected average, minimum and maximum purchase amount and purchase frequency of the customer for the on-demand service. Used for a more precise decision on payment method eligibility within the Klarna Payment flow for future payments.
<code>config.request_customer_token</code>	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 CONFIRMED



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,
  "payment_request_reference": "partner-payref-1234",
  "supplementary_purchase_data": {
    "ondemand_profile": {
      "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
    }],
  },
  "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:

```
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 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_request_reference": "partner-payref-1234",
  "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:

```
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 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"
}
```

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. 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:
 - line_items if the customer has added a subscription or for instance an insurance program with recurring charges to the basket
 - supplementary_purchase_data 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

 Support for subscription_items and supplementary_purchase_data 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_request_reference": "partner-payref-1234",
"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:

```

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 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_request_reference": "partner-payref-1234",
"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:

```

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 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_purchase_data": {
        "supplementary_details": {
          "insurance_details": {
            "insurance_company": "Insurance-4-U",
            "insurance_type": "CANCELLATION",
            "insurance_price": 1000
          }
        }
      }
    }
  ],
  "payment_request_reference": "partner-payref-1234",
  "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":

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"
}
```



2.6.2.3.3.3 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, include the customer token in the `X-Klarna-Customer-Token` header when creating a token charge request from your server. This parameter is included in the header to optimize the response time of the request. For a successful request, ensure the following parameters are included:

Parameter	Definition
<code>currency</code>	The currency in which the transaction is made.
<code>payment_amount</code>	The total amount to be charged.
<code>ondemand_profile.reference</code>	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.
<code>config.customer_present</code>	Set <code>customer_present</code> 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.
<code>config.redirect_url</code>	Share a <code>redirect_url</code> 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_eLZGI1B5dHBIRWcjZrNldnbEVj
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_request_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:

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": "CONFIRMED",
  "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_e1ZGI1B5dHBIRWcjZrN1dnbEVj
Klarna-Partner-Account: K12345
Content-Type: application/json

{
  "customer_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.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_request_reference": "partner-payref-1234",
"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 token charge 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_request_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": "CONFIRMED",
  "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	<p>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.</p> <p>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.</p> <p>The Partner can reattempt to charge the customer token again within 24 hours.</p>
200	DECLINED	PERMANENT	Permanent rejection. You must sign up the customer again.
200	SUBMITTED		<p>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.</p> <p>A distribution URL has been shared to redirect the user to the Klarna Payment flow to complete the payment.</p>

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

 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

 Support for management of customer tokens will be added in future releases.

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 v2/accounts/:accountId/identity/customer-token/introspect
POST v2/identity/customer-token/introspect

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj

{
  "customer_token": ""
}
```

Response example:

Response 201 OK

```
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 [v2/accounts/:accountId/identity/customer-profile/read](#)

Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj

```
{
  "customer_token": ""
}
```

Response example:

Response **200 OK**

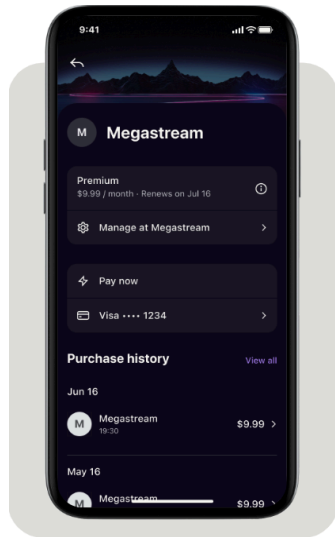
JavaScript

```
{
  "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"
    }
  }]
}
```

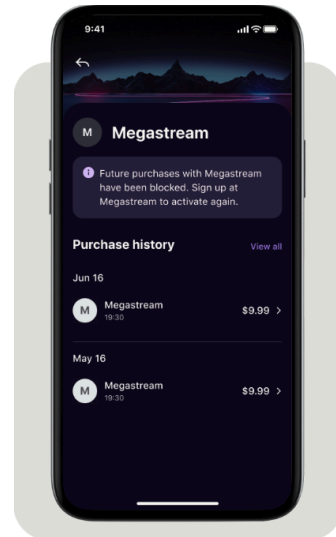
2.6.2.5.3 Cancel Customer Token

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 /v2/accounts/:accountId/identity/customer-token/revoke
POST /v2/identity/customer-token/revoke
Authorization: Basic klarna_live_api_e1ZGI1B5dHBIRWcjZrN1dnbEVj
{
  "customer_token": ""
}
```

2.6.2.5.4 Customer Token lifecycle webhook

Release notes

^{New!} 17 Support for webhooks for customer token lifecycle events will be added in future releases.

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](#) SUBMITTED 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 [/v2/accounts/{account_id}/payment/requests/{payment_request_id}](#).

⚠ This can only be performed when the payment request is in the **SUBMITTED** state. This restriction ensures that the 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_request_reference": "payment-reference-2024-05-27T12:50:42.677Z",
}
```

Response example:

```
JavaScript
{
  "state_context": {
    "distribution": {
      "url":
        "https://pay.test.klarna.com/eu/requests/b9bacf30-3619-60e5-bdcb-b0ad687d550b/start"
    }
  },
  "payment_request_reference": "payment-reference-2024-05-27T12:50:42.677Z",
  "payment_request_id": "krn:payment:eu1:request:b9bacf30-3619-60e5-bdcb",
  "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.mjs 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
}
```



2.6.3.2 Payments in mobile apps

 *Mobile app content to come in later releases*

2.6.3.3 Payments in physical store

 *Physical Store content to come in later releases*

2.6.3.4 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.

2.6.3.4.1 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 - card network classification	3000-3308, 4511	travel_reservations
Automobile/vehicle rentals - card network classification	3300-3499	travel_reservations
Hotels and motels - card network classification	3500-3999	lodging_reservations
Steamships and cruise lines	4411	travel_reservations
Airlines and air carriers - not elsewhere classified	4511	travel_reservations
Transportation services – not elsewhere classified	4789	travel_reservations
Marketplaces	5262	marketplace_seller_details
Lodging: Hotels, motels and resorts – Not else classified	7011	lodging_reservations
Timeshares	7012	lodging_reservations
Automobile rentals	7512	travel_reservations
Truck and utility trailer rentals	7513	travel_reservations
Theatrical producers (except motion pictures) and ticket agencies	7922	event_reservations
Bands, orchestras and miscellaneous entertainers – not elsewhere classified	7929	event_reservations
Amusement parks, circuses, carnivals and fortune tellers	7996	event_reservations
Aquariums, seaquariums, zoos and dolphinariums	7998	event_reservations



2.6.4 Important payment concepts

2.6.4.1 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

2021
17 Klarna performs ongoing assessments to ensure that address handling continues to meet security and operational needs.

2.6.4.1.1 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.

2.6.4.1.2 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.

2.6.4.1.3 Customer Data Requirements

2.6.4.1.3.1 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.

2.6.4.1.3.2 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

- [Li](#) (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](#).

2.6.4.2 Supplementary Purchase Data

Supplementary Purchase Data refers to additional data points that are not usually part of the basic requirements for a transaction to be placed. While these data points are technically optional, we strongly recommend that partners provide them to enhance underwriting and fraud assessment, as well as to improve the customer's post-purchase experience.

This information supports different use cases and may include data associated to specific industries or business models such as airline, marketplaces, as well as data related to the transaction (line items, subscriptions and on demand transactions) or even custom data.

The supplementary data shared with Klarna will support multiple use cases and add value in different fronts such as:

- **Effective Fraud Case Investigation:** Detailed transaction insights support thorough investigations, tailoring responses to unique characteristics in high-risk segments.
- **Klarna's Risk Exposure Monitoring:** Data points facilitate ongoing monitoring of risk exposure in high-risk segments, ensuring timely identification and control of potential risks.
- **Acceptance Rate Improvement:** Historical data informs underwriting processes, allowing for increased credit limits and improved acceptance rates.
- **Enhancement of solution offering:** Based on historical behavior of the consumer with a brand, Klarna is able to develop enhanced solutions for incentives and to drive particular actions.
- **Enhance post purchase experience:** Enhance customer's post-purchase experience with detailed transaction breakdowns, streamlining disputes and returns, reducing support requests, and boosting affiliate revenue through upselling and repurchasing opportunities—all within the Klarna app.
- **Enhance underwriting and fraud assessment processes:** enhance the underwriting and fraud assessment for partners and Klarna can make better-informed decisions in assessing the risk of the transaction. Historical data informs underwriting processes, allowing for increased credit limits and improved acceptance rates.

The following table list the supplementary_purchase_data data shared with Klarna will support multiple use cases and add value in different fronts such as:

Use Cases	Supplementary Purchase Data	Details
Travel services	travel_reservations	Applicable for any Partner that offers travel services including flights,bus, train, ferry. See specific MCC required to provide this information on Restricted businesses
Lodging services	lodging_reservation	See specific MCC required to provide this information on Restricted businesses



	s	
Car rental services	travel_reservations	Applicable for any Partner that offers car rental services. See specific MCC required to provide this information on Restricted businesses
Ticketing	event_reservations	Applicable for any Partner that offers tickets to access for events. See specific MCC required to provide this information on Restricted businesses
Coupons and voucher	voucher_details	Applicable for any Partner that offers to buy coupons or vouchers that are later exchanged for a service.
Insurance services	insurances	Applicable for any Partner that offers any type of standalone insurance policy.
Subscriptions	subscriptions	Applicable to any Partner that offers subscriptions to their customers. See details on Tokenized Payments
On-demand	ondemand_service	Applicable to any Partner that enables registration of the payment method to complete purchases faster on-demand. See details on Tokenized Payments
Registered checkout	customer_accounts customer_devices	Applicable for any segment, if a Partner supports registered checkout to complete a purchase.
Pick up in store	shipping	Applicable for any segment, if a Partner supports pick up in store for online purchases.
Shipping to multiple addresses	shipping	Applicable for any segment, if a Partner supports split of the products for delivery and to deliver in different addresses.
Selling in physical store	in_store_services - available in future releases	Applicable for any segment, if the transaction is done in a physical store.
Marketplace	marketplace_seller_details - available in future releases	Applicable when enabling marketplace services and processing transactions for sub-seller.
Line items	line_items	Applicable for any partners, represents essential detailed information such as the name, quantity, and unit price of a purchase.

2.6.4.2.1 Key values in Supplementary Purchase Data

2.6.4.2.1.1 The role of Line Items in Supplementary Purchase Data

Line items stand at the crossroads of Klarna's operations and are crucial in enhancing Klarna's fraud detection, underwriting capabilities, and customer experience. By breaking down transactions into individual units, they provide detailed insights that streamline dispute resolution and efficient interaction within the Klarna app, while reducing customer support errands

2.6.4.2.1.2 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.

2.6.4.2.2 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.

2.6.4.2.2.1 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	supplementary_purchase_data.< DATA OBJECT> shipping subscriptions customer ondemand_service	Send a structured json as part of the payment request creation operation.
Update payment	supplementary_purchase_data.< DATA OBJECT> shipping subscriptions customer ondemand_service	Send a structured json as part of the payment request update operation.

2.6.4.2.2.2 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.supplementary_purchase_data.<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
{
  ...
  {
    ...
    "supplementary_purchase_data": {
      "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"
    }
  ]
}
...

```

Example of a serialized json request


Unset

```

{ ...
  "supplementary_purchase_data": "{\\"content_type\\":\\"vnd.klarna.supplementary-data.v2\\",\\"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

 17 supplementary_purchase_data object will be available in future releases.

2.7 Interoperability

Interoperability refers to the seamless integration of Klarna's full product suite across the customer journey. The full potential of Klarna's services is realized when the complete offering is utilized. Given the multi-party dynamics within the Acquiring Partner constellation, achieving interoperability requires all parties to effectively integrate various aspects of Klarna's solutions.

Acquiring Partners must implement Klarna in a way that ensures all services and their related functionalities are either fully enabled or consistently accessible across all integration points. Following the steps in this document ensures compliance with the interoperability requirements outlined in Klarna's Network Rules for Acquiring Partners. This includes enabling key products such as On-Site Messaging (OSM), Express Checkout, and Sign in with Klarna.

2.7.1 Interoperability in practice

When a customer interacts with Klarna's other products, a session is initiated and maintained to ensure a seamless experience across various touchpoints. Klarna maintains the customers' payment capability within the session so that it is available at any time during the purchase session. Based on the payment capability, two properties are defined: **payment status** and **payment presentment instructions**.

- **Payment status** reflects how to proceed with the payment while
- **Payment presentment instructions** describe how to display Klarna in the payment selector.

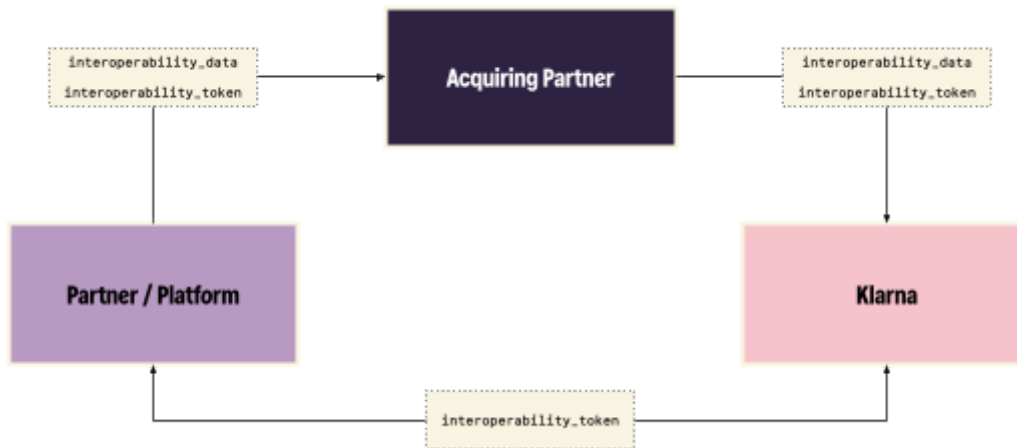
On any interaction with Klarna's other products, these values update to reflect the customer's commitment to pay with Klarna.

Example workflow:

1. **Session initiation:** At the start of the journey, the payment status is set to `REQUIRES_CUSTOMER_ACTION`, and presentment instructions are `SHOW_KLARNA`.
2. **Returning customer recognition:** When the customer is identified as a returning Klarna customer, the presentment instruction updates to `PRESELECT_KLARNA`.
3. **Express checkout flow:** During express checkout, the payment status may change to `PENDING_PARTNER_CONFIRMATION`, indicating that the Klarna payment request can be finalized directly without additional customer action.

To maintain continuity as the customer transitions to the Acquiring Partner's environment, the `interoperability_token` serves as a link for the customer session. This token enables Klarna to uphold a unified session, supporting personalization, keeping customers signed in across features, and enhancing checkout conversion by notifying you when a customer is ready to pay.





The `interoperability_token` is a JWT token signed by Klarna that contains all information needed to fulfill the seamless experience when transitioning through different integrations. Information of expiration is encoded in the token and will be taken care of by Klarna.

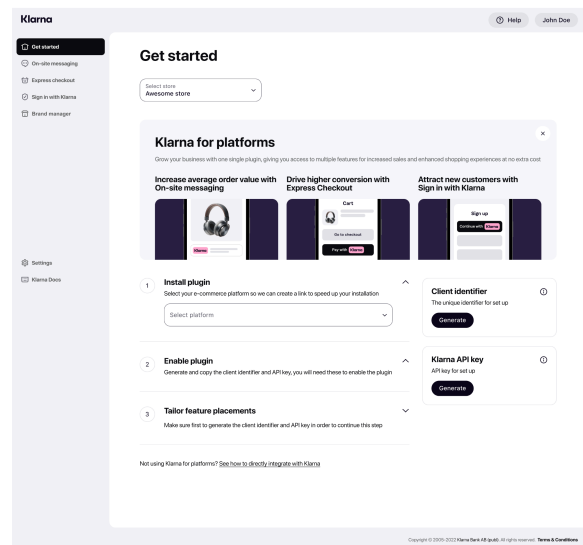
Together with the `interoperability_token`, Partners are also able to submit `interoperability_data` to increase acceptance rates by sharing customer data points or specific transactional data points.

2.7.1.1 Grant access to Klarna's Partner portal

Klarna provides access to non-payment products, such as Express Checkout, On-site Messaging, and Sign in with Klarna, through the Klarna Portal. Access enables Partners to retrieve credentials and manage their brand within Klarna's ecosystem. Acquiring Partners are required to grant access to the Klarna Portal to support these functionalities.

Key Features of the Klarna Portal:

- **Visual Assets:** Partners may upload logos, icons, social media URLs, and display names to control how they are presented to customers.
- **Boost Product Access:** Access to key conversion-enhancing features like Klarna Boost products is enabled through access to the portal.
- **Credential Management:** Partners are enabled to generate restricted credentials to supplement their Acquiring Partner integration.
- **Campaign Management:** Launch promotional campaigns (e.g., 0% financing offers).



⚠ Note: Partners cannot access features altering transaction statuses or privileged information, such as dispute or settlement management, unless otherwise enabled through their Acquiring Partner with Klarna.

Granting Portal Access

Access is managed through roles assigned collaboratively by Klarna and the Acquiring Partner. Multiple access methods are supported; however, the recommended approach must be leveraged wherever possible. Alternative solutions may be agreed upon with Klarna based on technical or business constraints.

The presentation of any link between your Partner-facing admin portal must be agreed and signed off in accordance with Klarna's user experience guidelines. Details are available in Klarna's user experience guidelines.

Recommended Solution

Signed [Deep Linking](#)

Use signed deep links secured with a JSON Web Token (JWT) to enable access without requiring password setup. Multi-factor authentication (MFA) is enforced on the Partner's portal.

Applicability: Suitable for all Acquiring Partners with a Partner-facing admin portal.

Alternate Solutions

Unsigned [deep linking](#) to Klarna Portal:

Relies on MFA enforced within the Klarna Portal and requires users to set up a password.

Applicability: For Partners unable to support JWT-signed deep linking due to Klarna-approved limitations.

Acquiring Partner as identity provider (SAML IdP):

Delegates authentication to the Acquiring Partner using SAML v2.0. No MFA is required on the Klarna Portal as MFA is managed by the Partner.

Applicability: Ideal for Partners with advanced technical capabilities to implement SAML IdP, providing a seamless user experience.

Klarna Portal user management API:

Grants API-based access for Partners without an admin portal to invite and manage users.

Applicability: Suitable for Partners without admin portal capabilities.

Release Notes

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

2.7.1.1 Deep linking

Deep linking facilitates seamless user access to Klarna's Partner Portal via either a signed or unsigned method.

When the user clicks a link that should redirect to the Klarna Portal from the your portal, the acquiring partner should use the Deep link API and redirect the User to Klarna using the url from the response. You need to share the email address of the User that will then need to create or log-in their Klarna Portal Account.

2.7.1.2 Recommended solution: Signed deep linking

By leveraging signed deep linking, the Acquiring Partner simplifies access to the Klarna Portal, avoiding password setup, supporting auditability of access, and ensuring non-repudiation. It can be enabled through the steps below:

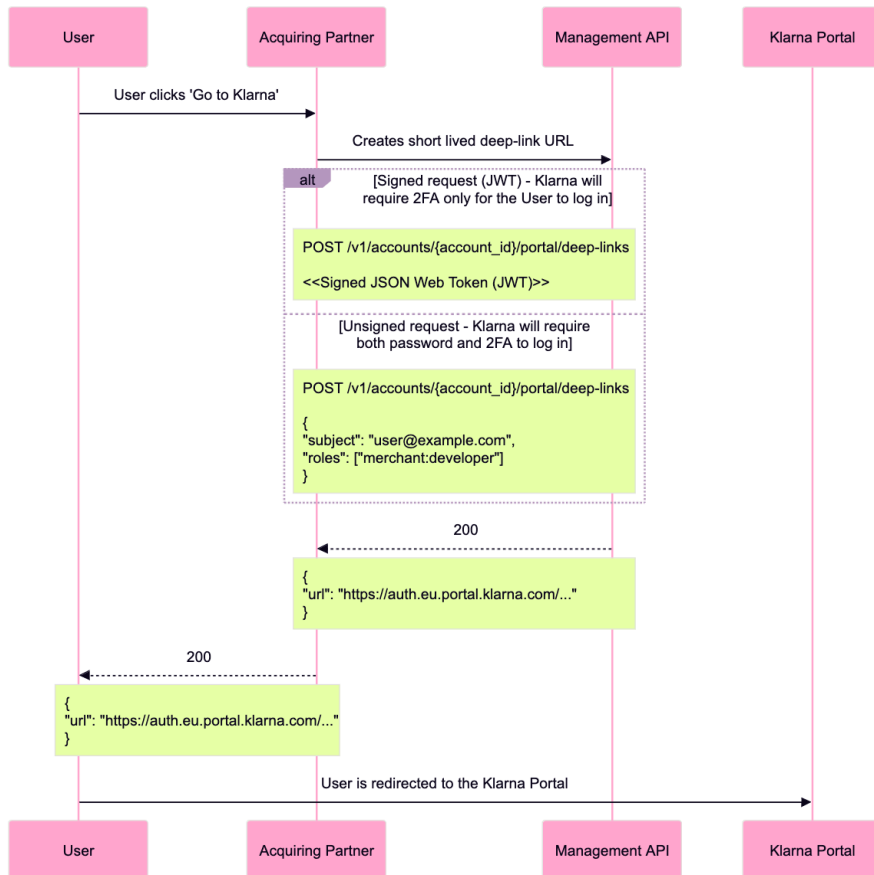
Step 1: Generate a JWT containing:



- User's email address (sub).
- Authentication details (amr).
- Account ID (account_id).
- Session token for redirection.

Step 2: Use Klarna's Deep Link API to generate a one-time URL.

Step 3: Redirect the user to Klarna Portal using the URL.



2.7.1.1.3 Alternate: Unsigned deep linking

Should you not be able to sign a JWT with the requested information, it is possible to use the Deep link API as a regular REST endpoint. In this case, the user would be forced to set a password when accessing the Klarna Portal leading to an additional step in the user journey which we would rather avoid. Both approaches are shown below.

2.7.1.1.1.1 Creating a deep link

Deep links for Partners are created using a POST request to [/v2/accounts/{account_id}/portal/deep-links](#) and are valid for one-time use, expiring after 60 seconds. Deep links should only be requested after the Partner expresses a desire to enter the Klarna Portal, and may only be generated for accounts onboarded via your services. The provisioned accesses remain valid until the user is logged out from Klarna.

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



- **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.

The deep-link API returns a URL that should be used to redirect the User to Klarna. Parameters vary depending on the usage of a signed JWT (definition 1 - ensures authentication delegation to the Acquiring Partner) or regular Payload (definition 2 - will require password setting by the User).

Definition 1: Content of the JWT that will be signed


Type	Parameter	Definition
Header	alg	An algorithm is used to generate the signature. Accepts "ES256" and "RS512".
	typ	Accepts "JWT" only.
	kid or x5c	Certificate serial number (kid) or the Certificate encoded (x5c).
Payload	iss	PSP's account ID, e.g. "krn:partner:global:account:live:LWT2XJSE".
	jti	Token ID. Must be unique per request
	sub	The email address of the user.
	iat	Time at which the JWT was issued. Either an integer or decimal, representing seconds past 1970-01-01 00:00:00Z, e.g. "1422779638".
	exp	Time at which the JWT expires. Either an integer or decimal, representing seconds past 1970-01-01 00:00:00Z, e.g. "1422779638".
	amr	Array of authentication methods used. Accepts "pwd" and "sms".
	account_id	ID of the sub-partner account, e.g. "krn:partner:global:account:live:LWT2XJSE".
	roles[]	Roles given to the User once they access the Portal. Roles give privileges for specific operations.
	session_token	When a Deep link is created due to the access to the Klarna User Access Provisioning URL, the session_token should be forwarded to Klarna. This session_token helps Klarna to redirect the User to the correct page they were looking for. It is a base64 token.
Signature	Computed signature following the JWT standard.	

Definition 2: Content of an unsigned Payload - will require users to create their own password

Parameter	Definition
subject	The email address of the user.
roles[]	Roles given to the User once they access the Portal. Roles give privileges for specific operations.
session_token	When a Deep link is created due to the access to the Klarna User Access Provisioning URL, the session_token should be forwarded to Klarna. This session_token helps Klarna to redirect the User to the correct page they were looking for. It is a base64 token.



Release notes

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

2.7.1.1.2 Revoking a deep link

Deep link access can be revoked at any time between its creation and session expiration, which occurs 8 hours after the link is generated.


Recommendations for Revocation

- Unused Links: Revoke a deep link if it is known it will not be utilized.
- Security Concerns: Revoke access if there is a need to terminate a user's ability to access the Klarna Partner Portal.

Deep links for Partners can be manually revoked using a DELETE request to /v2/accounts/{account_id}/portal/deep-links/{deep_link_id}

- {deep_link_id}: the unique identifier received when creating a deep-link.
- {account_id}: the account associated with the Partner.

Release notes

 Manually revoking a deep link endpoint will be made available in future releases.

2.7.1.1.2 Acquiring Partner as IdP

An Identity Provider (IdP) is a system that authenticates users and authorizes their access to applications and services. In this setup, The Acquiring Partner acts as the IdP and Klarna acts as the Service Provider (SP). By leveraging SAML v2.0, this setup ensures:

- Secure Authentication: User credentials remain securely managed by the IdP.
- Policy Enforcement: Enables security policies such as MFA.
- Enhanced Efficiency and Compliance: Centralized user management.

To enable Klarna Portal to leverage the Acquiring Partner as an IdP the following setup is required:

Step 1: Share SAML Configuration with Klarna

- Provide Klarna with the required SAML configuration details (e.g., metadata, certificates, endpoints).
- Klarna will complete the necessary configuration on their end.

Step 2: Set Up Klarna Portal as the Service Provider

- Configure Klarna Portal to accept SAML assertions from your IdP.
- Enforce MFA during SAML authentication for enhanced security.

Step 3: Map User Attributes

- Define and map user attributes from your IdP to the corresponding Klarna Portal users.
- Attributes may include partner account IDs, roles, and other relevant details.

Step 4: Handle SAML Authentication Requests

- Receive SAML authentication requests from Klarna.




- Process the requests in accordance with the SAML v2.0 protocol and return SAML authentication responses.

Step 5: Provision Access


- Use the Deep Link API or SAML Response to grant users access to the Klarna Portal.

Release notes

 17 the SAML IdP solution will be included in future releases. If you are interested in proceeding with this option for Partner Portal access please reach out to your Klarna representative to discuss further.

2.7.1.1.3 Klarna Portal user management API

Release notes

 17 Inviting and managing users will be further defined in later releases.

2.7.1.2 Educate Partners on Shopping Session API

The Shopping Session API enables Partners to share key data points directly with Klarna during the customer journey. This API supports edge cases where Partners may wish to or be required to provide additional data to Klarna regarding the customer or transaction, outside of the fields available via their Acquiring Partner

Data Handling: If both shopping and payment sessions are used, the data provided through the payment session will override the shopping session data.

No action is required by the Acquiring Partner to support this functionality. It is purely supplemental to the broader integration, however providing links to resources covering this functionality in your [public documentation for your Partners](#) is required to ensure optimal conversion rates for partners of all verticals.

2.7.2 User journeys enabled by Interoperability

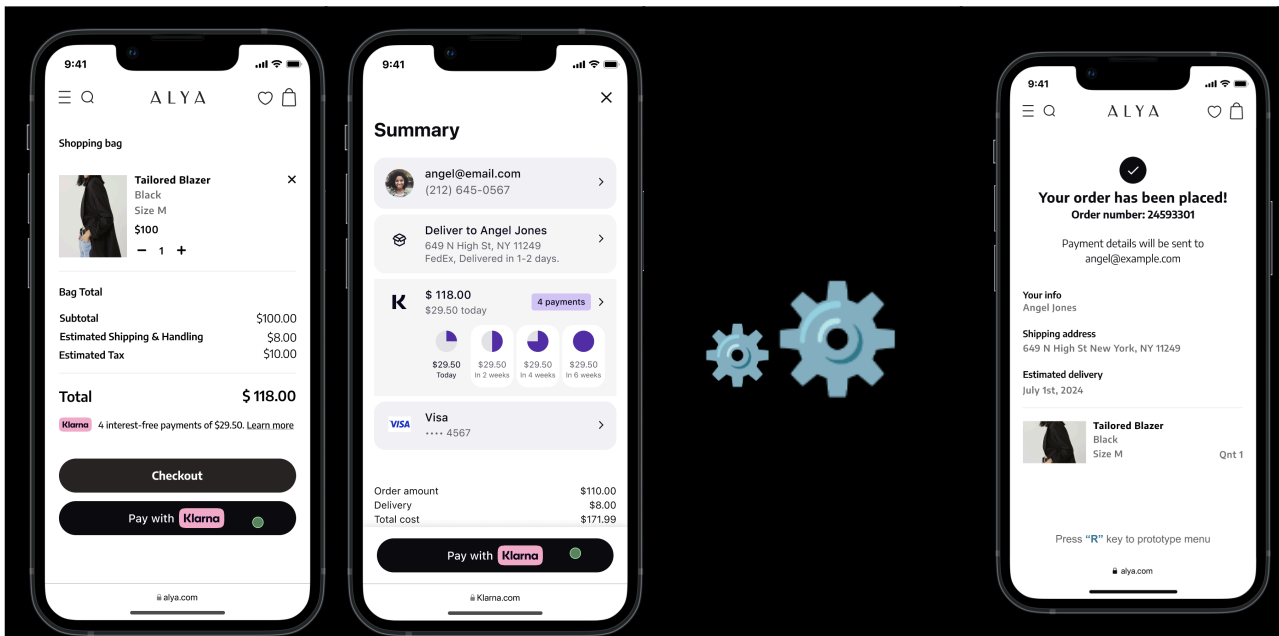
When properly implemented by each stakeholder in a purchase flow, interoperability unlocks key customer journeys that deliver a best-in-class user experience comparable to a direct integration. Below, we outline the targeted flows that should be supported, along with the expectations for each stakeholder.

2.7.2.1 Interoperability Flow 1: KEC 1-step

A Partner integrates Klarna Express Checkout feature. Once the customer completes the checkout flow, the Partner requests the Acquiring Partner to authorize the payment with Klarna. Klarna accepts the payment directly, provided the Acquiring Partner forwards the interoperability_token with a payment status of PENDING_PARTNER_CONFIRMATION.

KEC 1-step



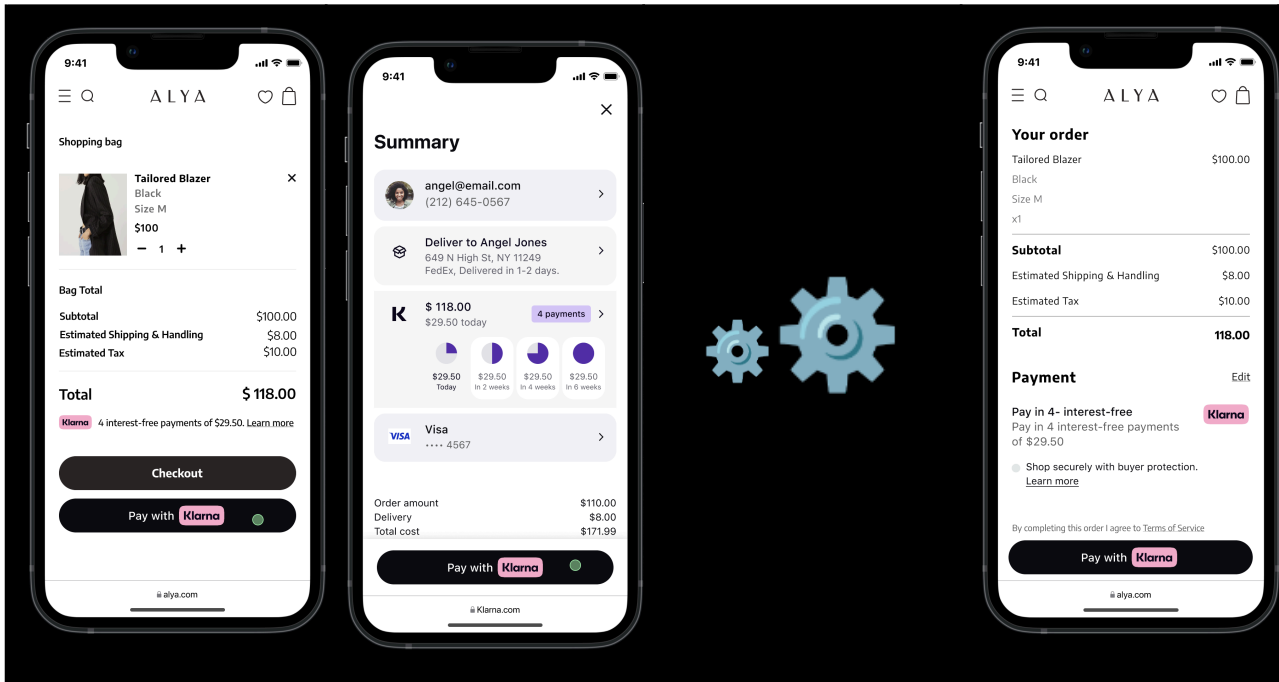


<p>Partner displays the Klarna Express Checkout button by directly integrating Klarna's Web SDK.</p>	<p>Consumers are able to select their shipping option and confirm their payment with Klarna.</p>	<p>Partner registers a Payment with the Acquiring Partner asking for the payment to be directly finalized. The interoperability_token is shared to the Acquiring Partner.</p>	<p>As the final payment amount and order lines are accepted (and have not changed), the payment is directly validated without further interaction.</p>
<p>The consumer's payment status is REQUIRES_CUSTOMER_ACTION.</p>		<p>The consumer's payment status is PENDING_PARTNER_CONFIRMATION.</p>	<p>N/A</p>

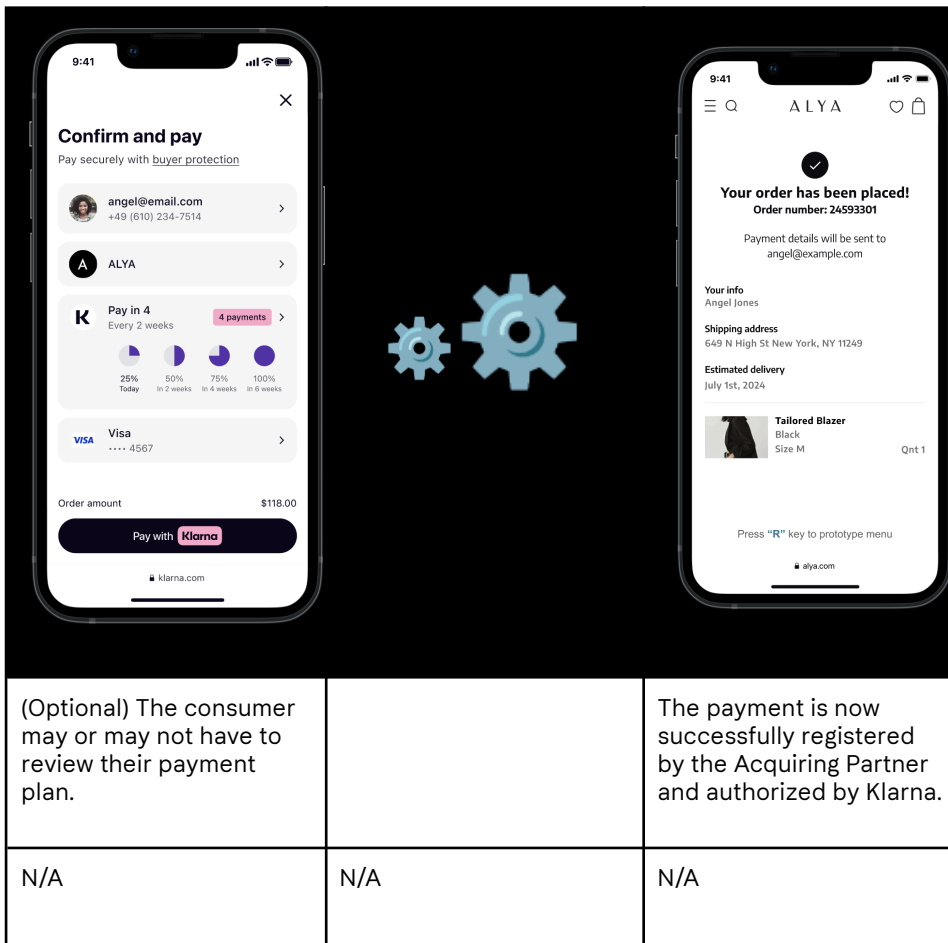
2.7.2.2 Interoperability Flow 2: KEC 1-step requiring step-up

A Partner integrating Klarna Express Checkout directly but without the support for dynamic shipping options selection. After completing the Express flow, the Partner requests the Acquiring Partner to authorize the payment with Klarna. Due to the adjusted shipping amount causing the order total to exceed the approved amount, Klarna rejects the payment when the Acquiring Partner forwards the interoperability_token, and responds with a step-up flow. The customer is then presented with a checkout confirmation screen, where they confirm the adjusted payment with Klarna.

KEC 1-step where step-up is required due to order amount mismatch



<p>Partner displays the Klarna Express Checkout button by directly integrating Klarna's Web SDK.</p>	<p>The customer gets approved by Klarna.</p>	<p>Partner registers a Payment with the Acquiring Partner asking for the payment to be directly finalized. The <code>interoperability_token</code> is shared with the Acquiring Partner.</p> <p>As the payment amount has changed (or similar error where previous authorization is not valid anymore), the Payment Request requires the consumer to approve again.</p>	<p>The Acquiring Partner, when owning the payment selector, shows only Klarna and hides all other payment options, as indicated by the presentment instruction <code>SHOW_ONLY_KLARNA</code>.</p> <p>When clicking the Pay with Klarna button, the customer may go through the Payment Flow to approve the final payment.</p>
<p>The consumer's payment status is <code>REQUIRES_CUSTOMER_ACTION</code>.</p>		<p>The consumer's payment status is <code>PENDING_PARTNER_CONFIRMATION</code>.</p>	<p>The consumer's payment status is <code>REQUIRES_CUSTOMER_ACTION</code>.</p> <p>The presentment instruction is <code>SHOW_ONLY_KLARNA</code>.</p>



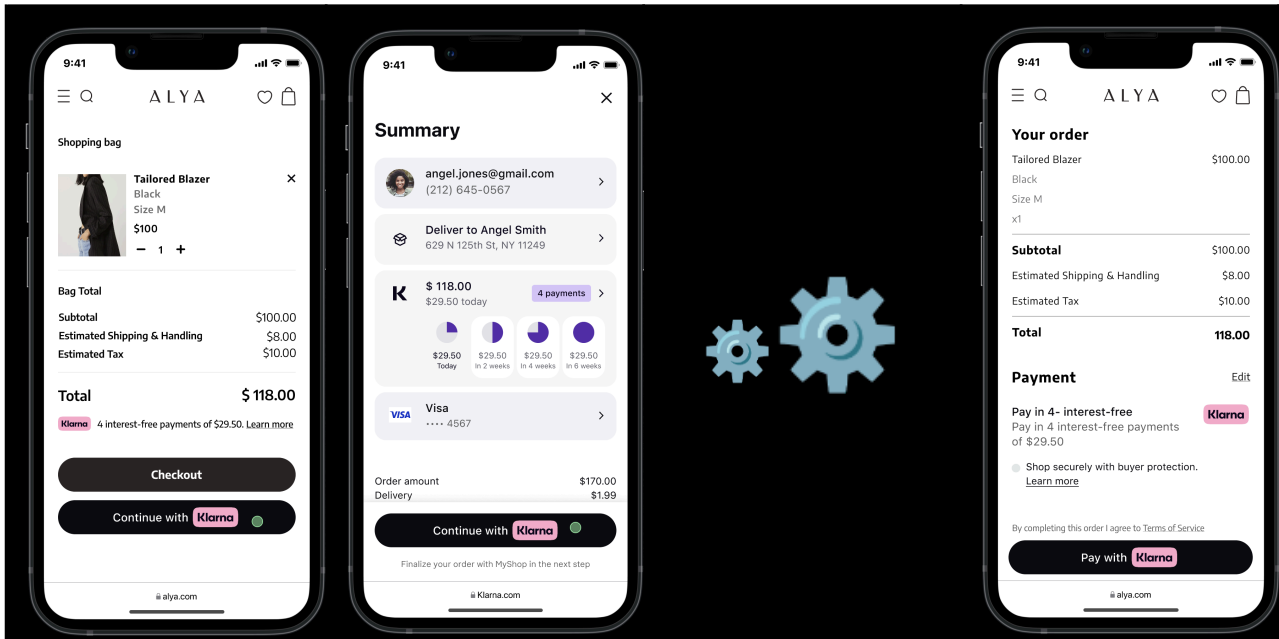
2.7.2.3 Interoperability Flow 3: KEC multistep

A Partner integrating Klarna directly into their checkout flow as a conversion booster supports express checkouts with a two-step process:

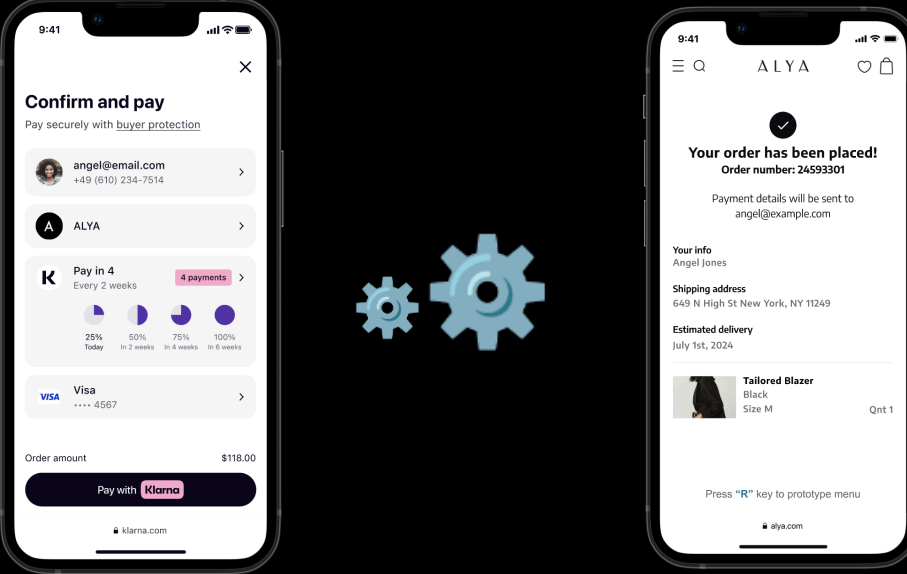
1. Customers first select their shipping address with Klarna.
2. They approve their purchase as a second step.

After completing these steps, customers are presented with a checkout confirmation screen, where they can validate the payment with Klarna. The Partner forwards the `interoperability_token` to the Acquiring Partner when initiating the checkout process.





<p>Partner displays the Klarna Express Checkout button by directly integrating Klarna's Web SDK.</p>	<p>The customer gets pre-approved by Klarna.</p>	<p>Partner initiates a Checkout session with the Acquiring Partner. The <code>interoperability_token</code> is shared to the Acquiring Partner.</p>	<p>The Acquiring Partner, when owning the payment selector, preselects Klarna and hides all other payment options, as indicated by the presentment instruction <code>SHOW_ONLY_KLARNA</code>.</p> <p>When clicking the Pay with Klarna button, the customer may go through the Payment Flow to approve the final payment.</p>
<p>The consumer's payment status is <code>REQUIRES_CUSTOMER_ACTION</code>.</p>			<p>The consumer's payment status is <code>REQUIRES_CUSTOMER_ACTION</code>. The presentment instruction is <code>SHOW_ONLY_KLARNA</code>.</p>



<p>(Optional) The consumer may or may not have to review their payment plan in case the amount variation is above a specific threshold (country and payment option dependant).</p>	<p>Partner registers a Payment with the Acquiring Partner asking for the payment to be directly finalized.</p>	<p>The payment is now successfully registered by the Acquiring Partner and authorized by Klarna.</p>
<p>N/A</p>	<p>N/A</p>	<p>N/A</p>

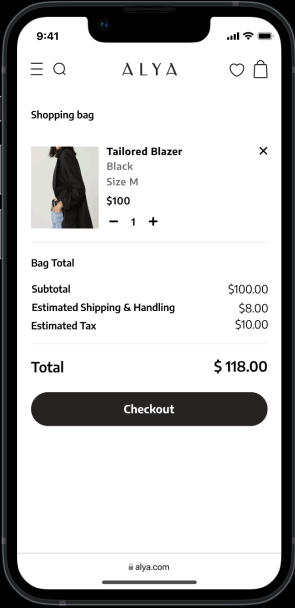
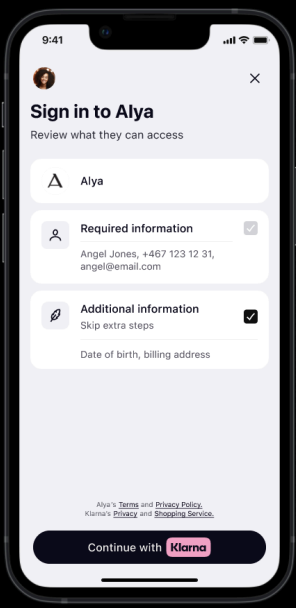
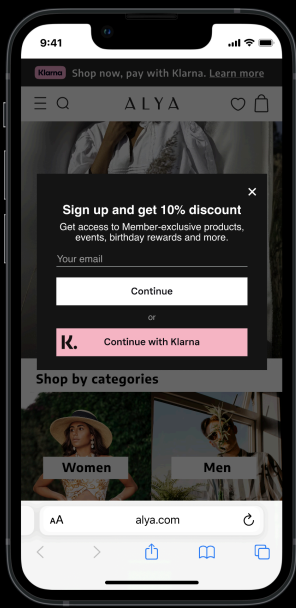
2.7.2.4 Interoperability Flow 4: Sign in with Klarna

A Partner integrating Sign in with Klarna allows customers to log in to their website using their Klarna account. Once the customer completes the checkout process, they validate the payment with Klarna. The Partner forwards the interoperability_token to the Acquiring Partner when initiating the checkout process.

In this scenario, since the customer has already indicated their intent to pay with Klarna by logging in, the Acquiring Partner should:

- Preselect Klarna in the payment selector.
- Minimize visibility of other payment options to optimize conversion rates.

Sign in with Klarna



Partner displays Sign-In with Klarna button by directly integrating Klarna's Web SDK.

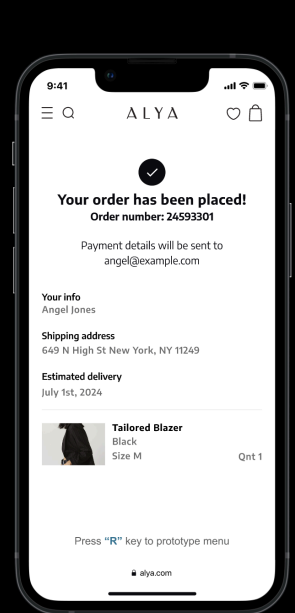
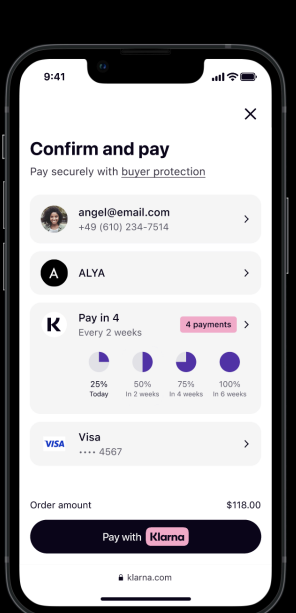
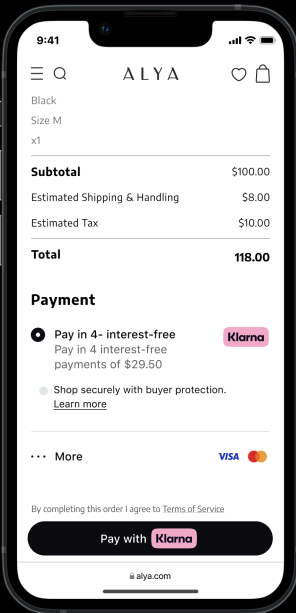
Customer logs in with their Klarna credentials.

The customer proceeds with their order.

Partner initiates a Checkout session with the Acquiring Partner. The `interoperability_token` is shared to the Acquiring Partner.

The consumer's payment status is `REQUIRES_CUSTOMER_ACTION`.

The consumer's payment status is `REQUIRES_CUSTOMER_ACTION`. The presentment instruction is `PRESELECT_KLARNA`.



The Acquiring Partner, when owning the payment selector, preselects Klarna and collapses all other payment options, as indicated by the presentment instruction PRESELECT_KLARNA.	(Optional) The consumer may or may not have to review their payment plan in case the amount has changed too much.	The payment is now successfully registered by the Acquiring Partner and authorized by Klarna.
The consumer's payment status is REQUIRES_CUSTOMER_ACTION. The presentment instruction is PRESELECT_KLARNA.	N/A	N/A

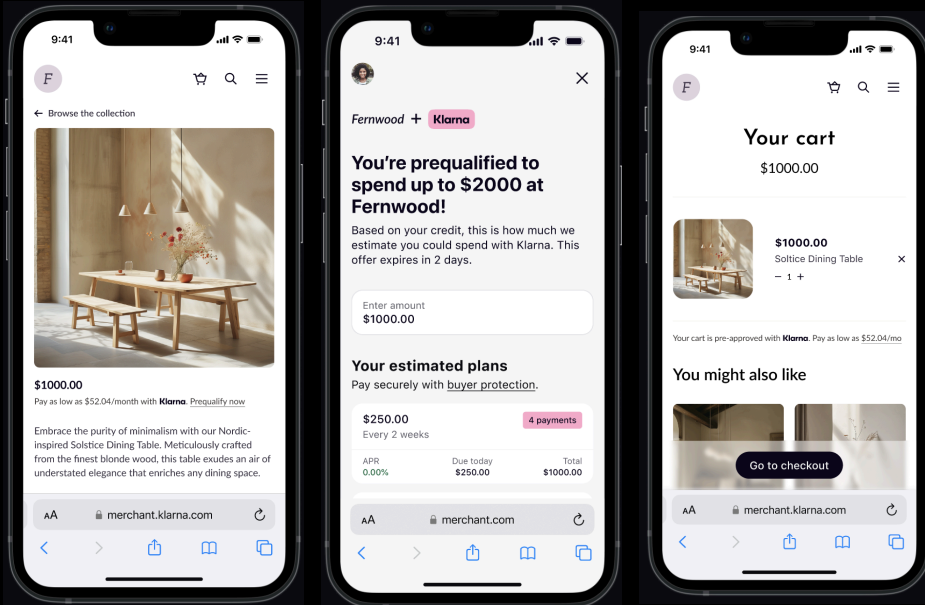
2.7.2.5 Interoperability Flow 5: Pre-qualification

A Partner integrates Klarna On-site Messaging which allows customers to check their credit eligibility prior to entering the checkout process. After receiving a pre-qualification amount, the customer proceeds through checkout and validates the payment with Klarna. The Partner forwards the `interoperability_token` to the Acquiring Partner when initiating the checkout.

In this scenario:

- If the customer is pre-qualified and has indicated their intent to pay with Klarna, Klarna will share the following instructions to the Acquiring Partner:
 - Preselect Klarna in the payment selector.
 - Minimize visibility of other payment options to optimize conversion rates.
- If the customer is not approved for pre-qualification, Klarna will not provide instructions to preselect Klarna.

Pre-qualification



Partner displays an On-site messaging with "Prequalify now" button directly integrating Klarna's Web SDK.

The consumer checks their availability by interacting with the messaging element.

The customer gets pre-approved by Klarna.

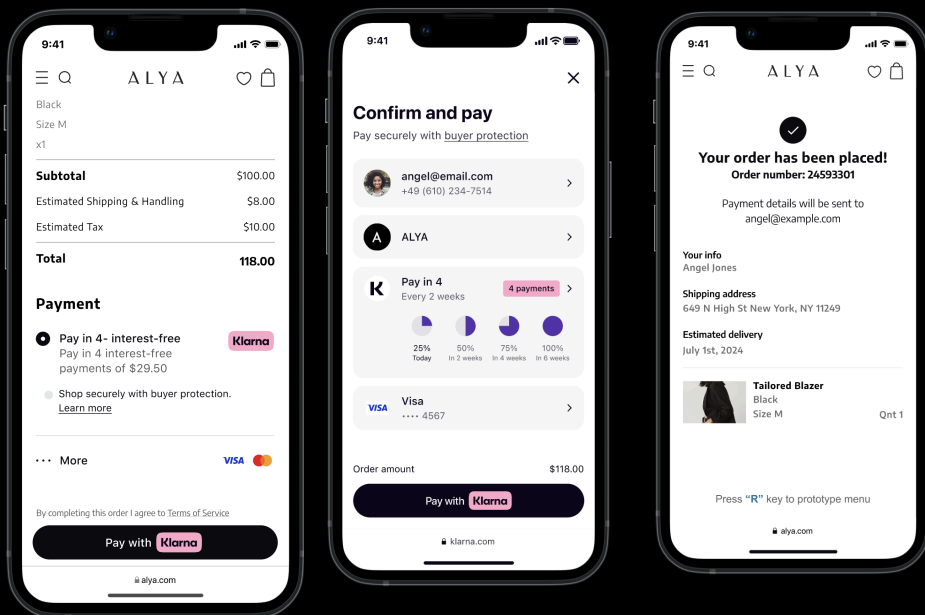
Partner initiates a Checkout session with the Acquiring Partner. The `interoperability_token` is shared to the Acquiring Partner.

The consumer's payment status is `REQUIRES_CUSTOMER_ACTION`.

The presentment instruction is `SHOW_KLARNA`.

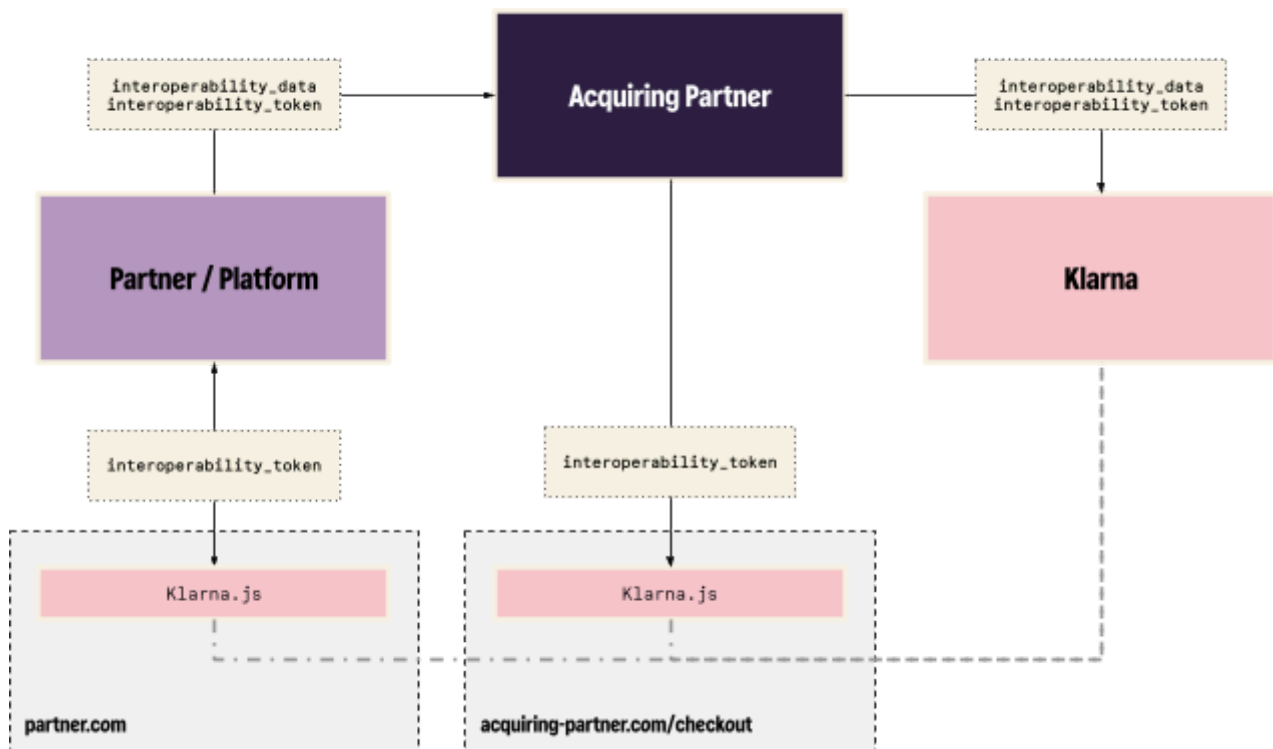
The consumer's payment status is `REQUIRES_CUSTOMER_ACTION`.

The presentment instruction is `PRESELECT_KLARNA`.



The Acquiring Partner, when owning the payment selector, preselects Klarna and collapses all other payment options, as indicated by the presentment instruction PRESELECT_KLARNA.	(Optional) Consumers may or may not have to review their payment plan should the order amount have increased or the payment method needs to be re-selected.	The payment is now successfully registered by the Acquiring Partner and authorized by Klarna.
The consumer's payment status is REQUIRES_CUSTOMER_ACTION. The presentment instruction is PRESELECT_KLARNA.		N/A

2.7.3 Concepts behind payment interoperability



2.7.3.1 Payment status

Payment status provides information on a likely outcome of the payment request when submitting the `interoperability_token`. It's not required to base integration logic on the payment status, but it's a clear indication on what to expect and is fundamental to understanding the interoperability requirements.

Payment Status	Use Cases	Description	Integration Actions
PENDING_PARTNER_CONFIRMATION	KEC	The customer has approved the purchase.	Create a payment request if Klarna was selected.

REQUIRES_CUSTOM ER_ACTION (default)	SIWK, Pre-qualification, Standard checkout	The customer needs to approve the purchase.	The payment request can be returned with the CONFIRMED state immediately. If step up is required, the state will be SUBMITTED, in which case, the integrator is supposed to launch the Klarna purchase flow.
UNSUPPORTED	SIWK	Klarna is not offered for the current payment context.	Reject the merchant request if Klarna was selected. If a Klarna payment request is submitted, it won't be rejected, the customer may enter the purchase flow but won't be able to complete it.

Simply create a payment request and pass the token to Klarna, the payment request's state will indicate what to do next. If your integration is not creating payment requests when receiving the `interoperability_token`, then you will most likely need to adapt your logic to the payment status.


⚠ For a future proof solution, an unrecognized status should be considered as `REQUIRES_CUSTOMER_ACTION`. This would allow Klarna to add more statuses in the future and not break your integration.


2.7.3.2 Payment presentment instruction

In case a payment selector is displayed, the integrator is required to adhere to Klarna's payment presentment instruction to achieve the best-in-class user experience.

It is important to note, that presentment instruction is not connected directly to the Payment Status, but a Payment Status `PENDING_PARTNER_CONFIRMATION` will always be associated to `SHOW_ONLY_KLARNA`.

Presentment Instruction	Use Cases	Description	Integration Actions	Payment Selector UX
SHOW_ONLY_KLARNA	KEC, Prequalification	The customer has already approved the purchase or shown clear intent for paying with Klarna.	The payment selector must show only Klarna and collapse all other options with a UX pattern implying that no choice needs to be made as it was done previously. Other options can be uncollapsed if the purchase with Klarna fails.	
PRESELECT_KLARNA	SIWK	The customer is recognized by Klarna and likely to pay with Klarna.	The payment selector Klarna must be pre-selected and put first as conversion will be increased. Klarna suggests collapsing all other payment methods.	

Presentment Instruction	Use Cases	Description	Integration Actions	Payment Selector UX
SHOW_KLARNA (default)	Standard checkout	The customer is not recognized by Klarna (yet).	Klarna should be available in the payment selector, respecting the ordering of the merchant and the Klarna guidelines.	<p>Payment</p> <p><input type="radio"/> Pay now Pay in full today Klarna</p> <hr/> <p><input type="radio"/> Pay in 4- interest-free Pay in 4 interest-free payments of \$29.50 Klarna</p>
HIDE_KLARNA	Standard checkout	Klarna is not offered for the current payment context, due to currency not being supported for example.	Don't show a Klarna button or offer Klarna.	<p>Payment</p> <p><input type="radio"/> Credit card VISA </p>

Presentment Instruction	Use Cases	Payment Selector UX
SHOW_ONLY_KLARNA	KEC, Prequalification	<p>Payment Edit</p> <p><input checked="" type="radio"/> Pay in 4- interest-free Pay in 4 interest-free payments of \$29.50 Klarna</p> <p><input type="radio"/> Shop securely with buyer protection. Learn more</p>
PRESELECT_KLARNA	SIWK	<p>Payment</p> <p><input checked="" type="radio"/> Pay in 4- interest-free Pay in 4 interest-free payments of \$29.50 Klarna</p> <p><input type="radio"/> Shop securely with buyer protection. Learn more</p> <p>... More VISA </p>
SHOW_KLARNA (default)	Standard checkout	<p>Payment</p> <p><input type="radio"/> Pay now Pay in full today Klarna</p> <hr/> <p><input type="radio"/> Pay in 4- interest-free Pay in 4 interest-free payments of \$29.50 Klarna</p>
HIDE_KLARNA	Standard checkout	<p>Payment</p>

Presentment Instruction	Use Cases	Payment Selector UX
		<input type="radio"/> Credit card 

2.7.3.3 Interoperability data points

Data point	Content	How to accept from Partners	When to share with Klarna
interoperability_token	An encoded token that will allow Klarna to ensure consistency of the consumer journey.	<p>Allow Partners to forward the interoperability_token on all APIs involved in accepting a Payment, tokenizing a Consumer or starting a Checkout flow.</p> <p>Token expiration duration is decided by Klarna. A partner should always be able to send a new token for the same session. Invalid or expired tokens will be ignored by Klarna but not rejected to ensure resilience of the API. The Acquiring Partner does not need to validate the tokens.</p>	<p>Share with Klarna on all server-side calls.</p> <p>Share with Klarna when initiating a client-side journey with the Web SDK.</p>
interoperability_data	A serialized JSON respecting the Interoperability Data Schema set by Klarna to allow Partners to send extra data points for better conversion rates and consumer experience.	<p>Allow Partners to forward the interoperability_data on all APIs involved in accepting a Payment, tokenizing a Consumer or starting a Checkout flow.</p> <p>Allow Partners to forward interoperability_data on all APIs involved in the post-purchase operations such as captures and refunds.</p> <p>Client-side integration should not support interoperability_data as the data sent by partners may be sensitive.</p>	Share with Klarna on all server-side calls that support it.

2.7.4 Interoperability requirements

In order to achieve true feature parity for the Klarna product suite, we have identified the following key requirements:



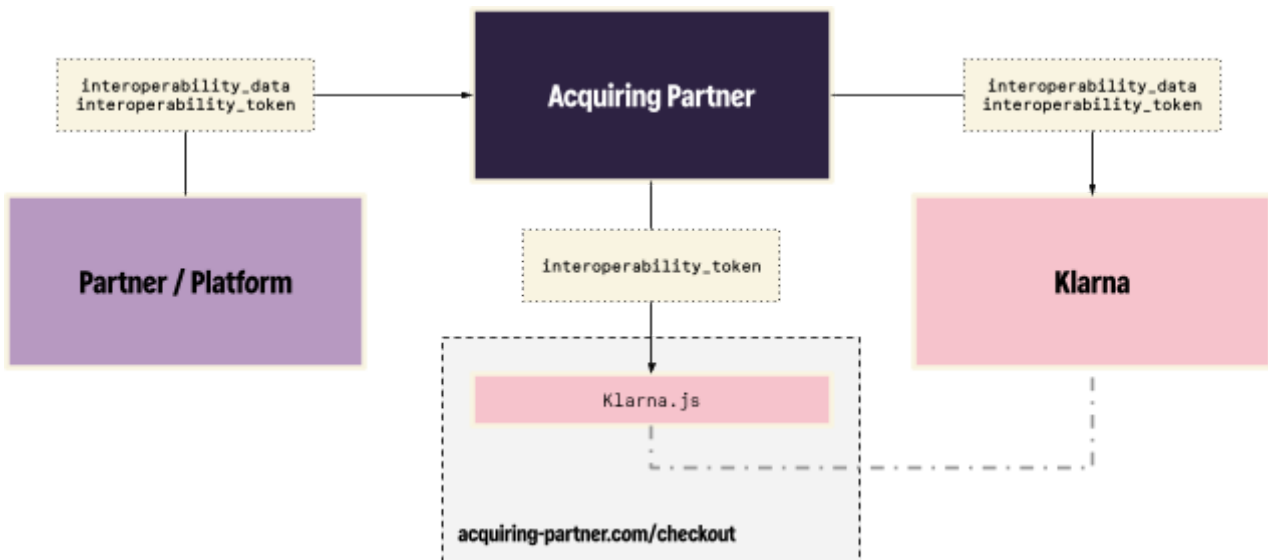
Requirement #	Description	How to achieve
Requirement 1.1 Accept Interoperability data points	A Partner must be able to send a Klarna Interoperability Token in all APIs used to interact with Klarna.	Cookbook: Accept interoperability data points from the Partner through Payment & Checkout APIs
Requirement 1.2 Forward Interoperability data points	The Acquiring Partner must forward the interoperability token to respective Klarna APIs whenever they receive one.	Cookbook: Acquiring Partner forwards the interoperability_token when interacting with Klarna
Requirement 2 Follow payment presentation instructions when displaying payment method selector	When a Partner renders a payment selector built by the Acquiring Partner (examples: embeddable components, hosted checkout) with an interoperability_token, respect the payment presentation instruction to maximize customer experience and conversion.	Cookbook: Build the payment selector
Requirement 3 Provide presentment instruction in APIs returning available payment methods	If the Acquiring Partner has APIs that sends back information about available payment methods, the description of each Klarna payment method should be retrieved from Klarna Payment Descriptor API and a payment selector presentment instruction should be included in the response.	Cookbook: Support payment presentment instructions in APIs returning available payment methods
Requirement 4 Handles Klarna payments with or without customer interactions	A Partner must be able to use a Klarna Interoperability Token to create a Klarna payments transaction without customer interaction. Note: if the Acquiring Partner API provides a "step up flow", then Klarna's "step up flow" must be available.	Cookbook: Accept a Payment with Klarna Cookbook: Accept a Payment with Klarna requiring partner confirmation
Requirement 5 Accept Interoperability Token to register an offline payment method	A Partner must be able to receive a recurring payment token by submitting an Interoperability Token without customer interaction. Note: if the Acquiring Partner API provides a "step up flow", then Klarna's "step up flow" must be available.	Cookbook: Create payment token based on interoperability_token for recurring payments
Requirement 6 Accept and forward Interoperability Data for post-purchase API	A Partner must be able to send Klarna Interoperability Data in all post-purchase operations	Cookbook: Retrieve and pass interoperability on post-purchase APIs



2.7.5 Cookbooks to achieve requirements

2.7.5.1 Cookbook: Accept interoperability data points from the Partner through Payment & Checkout APIs

Context: The interoperability data points [interoperability_token, interoperability_data] ensure all information and context associated with a payment transaction is preserved throughout the journey of a customer. The specific requirements for applying interoperability depend on the chosen integration path, ensuring a seamless customer experience.



To enable Partners to forward those data points, Acquiring Partners must support them in their respective API endpoints which allows Partners to initiate a payment or checkout flow.

All integration patterns must support the `interoperability_token`, while only server-side operations should support `interoperability_data`.

2.7.5.1.1 Enabling server-side sharing of the interoperability data points

Acquiring Partners must accept Klarna's `interoperability_token` and `interoperability_data` at all touchpoints where payments or checkout are initiated through server-side API endpoints. For example:

```
Java
POST api.acquiring-partner.com/v2/payments
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "interoperability_data": "...
    }
  }
}
```

⚠ The Partner-facing field should be named either `klarna_interoperability_token` or `interoperability_token` in a Klarna-specific context, ensuring that it's easy for any Partner to identify and use. While the Acquiring Partner can introspect the token, they must not modify it and must always forward it.

2.7.5.1.2 Enabling client-side sharing of the Klarna Interoperability Token

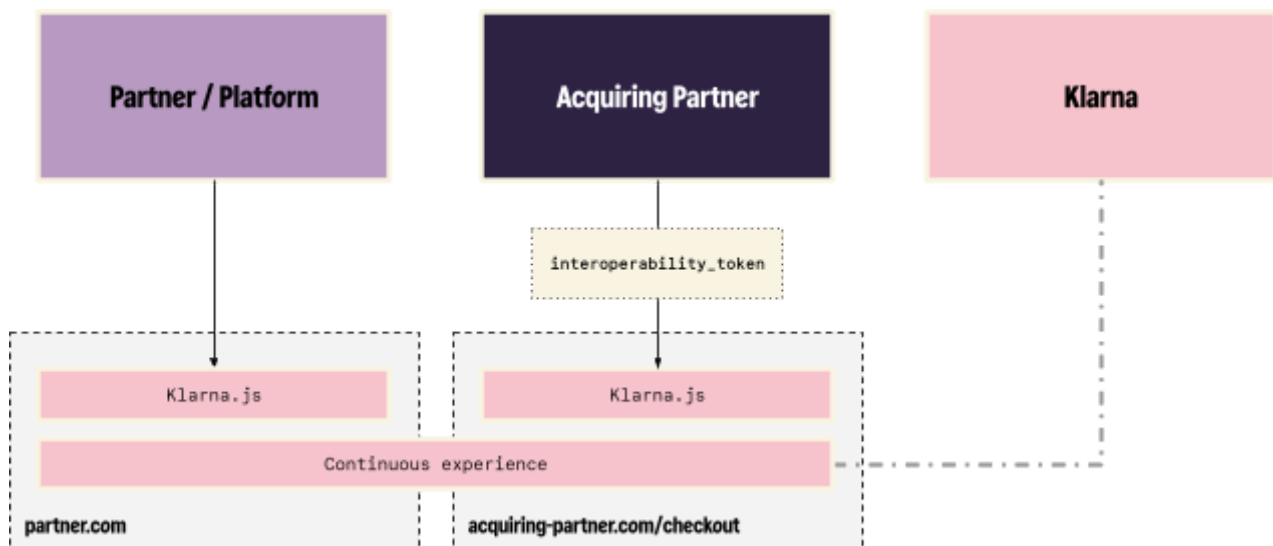
In cases where Acquiring Partner offers payment or checkout initiation flow using a client-side API (e.g. javascript library), Acquiring Partner must accept Klarna's `interoperability_token` at respective endpoints. For example:

```
JavaScript
acquiringPartnerLib.initiatePayment({
  currency: "USD",
  amount: 17800,
  paymentMethodOptions: {
    klarna: {
      interoperabilityToken: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
  }
})
```

The `interoperability_token` should not be accepted, as it may contain sensitive data.

2.7.5.2 Cookbook: Resume interoperability on Acquiring Partner domain via Web SDK

Context: When Partner provides the `interoperability_token` to Acquiring Partner, or it's retrieved by Acquiring Partner in a Platform/Plugin scenario, the token must be used to resume interoperability when initiating the Web SDK.



It is required to enable continuation of a customer journey and interoperability on an Acquiring Partner's domain (e.g. hosted checkout page). To achieve this, use the Klarna SDK initialization method:

```

JavaScript
<script type="module">
  const { KlarnaSDK } = await import("https://js.klarna.com/web-sdk/v2/klarna.mjs")

  const Klarna = await KlarnaSDK({
    clientId: "[client-id]",
    accountId: "[account-id]",
    interoperabilityToken: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  })

  // Klarna SDK ready to be utilized and session is restored
  // Klarna.Payment.button().mount('#payment_container')

  // if interoperabilityToken can't be set while initializing the KlarnaSDK
  Klarna.Configuration.setInteroperabilityToken("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...")
</script>

```

This will ensure that the customer journey is continued on the Acquiring Partner's domain and interoperability is achieved on Klarna touch-points (e.g. personalized On-Site Messaging or Payment Selector). This method invocation should be done before calling any other methods within the Web SDK. Any further method calls using Web SDK will implicitly utilize this token to ensure continuity of the session.

2.7.5.3 Cookbook: Acquiring Partner forwards the Interoperability data points when interacting with Klarna

Context: When Partner provides interoperability data points [interoperability_token, interoperability_data] to Acquiring Partner, or it's retrieved by Acquiring Partner in a Platform/Plugin scenario, the data points must be shared with Klarna.

Acquiring Partners must forward the interoperability_token in respective Klarna APIs (e.g. Payment API, Messaging API) when they pursue a server-side integration. In Payment APIs and Messaging APIs, it is mandatory for achieving the continuity of customer journey and for receiving personalized messaging content for the specific customer (e.g. deals and discounts).

For example, when creating a Payment Request, interoperability_token and interoperability_data must be forwarded as below:

```

Java
POST api-global.klarna.com/v2/accounts/{account_id}/payment/requests
{
  "currency": "USD",
  "payment_amount": 17800,
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

```

In all other APIs, the data points may be independently expected from endpoints. For example, the payment presentment API is not expecting interoperability_token.

```

Java
POST
https://api-global.klarna.com/v2/accounts/{account_id}/payment/messaging/payment-presentment
{
  "locale": "en-US",
  "payment_amount": 1000,
  "category_preference": "KLARNA",
  "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}

Response:
{
  "instruction": "SHOW_KLARNA",
  "payment_status": "REQUIRES_CUSTOMER_ACTION",
  "presentment_options": [
    { "payment_option_id": "cGF5bWVudC1..NDg4ODI=", "preselected": true},
    { "payment_option_id": "cGF5bWVudC1..ND34deWs", "preselected": false}
  ],
  "payment_descriptors": [{...}],
  "impression_url": "https://..."
}

```

2.7.5.3.1 Token validation and expiry

Klarna will not reject requests if tokens are invalid or expired. Instead, the tokens are simply ignored in these cases. This ensures resilience of the API in the context where the Acquiring Partner is not the one generating the tokens.

2.7.5.4 Cookbook: Retrieve the Payment Status and Presentment Instruction

To be able to comply with requirements, the Acquiring Partner may need to know the payment status and presentment instructions. This can be achieved by following one of the solutions listed below:

2.7.5.4.1 Read payment presentment via Web SDK

Once the interoperability token is provided inside Web SDK configuration, the Acquiring Partner can use Klarna Web SDK `Klarna.Messaging.presentment()` method to read presentment instructions:

```

JavaScript
const Klarna = await KlarnaSDK({
  //..
  interoperabilityToken: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
})

// Ensure that this is called after the interoperabilityToken is set
const presentment = await Klarna.Messaging.presentment()
const presentmentInstruction = presentment.instruction
const paymentStatus = presentment.paymentStatus

```

Query payment presentment from Klarna API

Alternatively a direct API call can be made to fetch the presentment instructions together with payment descriptors from Klarna Payments API by submitting the `interoperability_data`:



```

Java
POST
https://api-global.klarna.com/v2/v2/accounts/{account_id}/payment/messaging/payment-presentment
{
  "locale": "en-US",
  "payment_amount": 1000,
  "category_preference": "KLARNA",
  "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}

Response:
{
  "instruction": "SHOW_KLARNA",
  "payment_status": "REQUIRES_CUSTOMER_ACTION",
  "presentment_options": [
    { "payment_option_id": "cGF5bWVudC1..NDg4ODI=", "preselected": true},
    { "payment_option_id": "cGF5bWVudC1..ND34deWs", "preselected": false}
  ],
  "payment_descriptors": [{...}],
  "impression_url": "https://..."
}

```

2.7.5.5 Cookbook: Build a payment selector respecting presentment instructions

Context: The Partner is creating a checkout session using the Acquiring Partner's APIs. The Acquiring Partners forward the `interoperability_token` to Klarna and are expecting to show a payment selector. The payment selector is built by the Acquiring Partner.

To shorten checkout times and increase conversion, we require specific presentation of Klarna based on the `presentment_instructions`. This status is modified through the different interactions the customer has in the Partner domain. The expected layout of the payment selector is described in [Presentment instruction](#).

To render a payment selector, the usual steps would be:

1. Initialize Klarna JS using the `interoperability_data` received from the Partner,
2. Use the `presentment()` Web SDK method to retrieve the presentment instructions and descriptor assets,
3. Present the payment selector according to the presentment instructions.

```

JavaScript
<script type="module">
  const { KlarnaSDK } = await import("https://js.klarna.com/web-sdk/v2/klarna.mjs")

  const Klarna = await KlarnaSDK({
    clientId: "[client-id]",
    accountId: "[account-id]",
    interoperabilityToken: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  })

  const presentment = await Klarna.Messaging.presentment()
</script>

```


2.7.5.1 When instruction is to show only Klarna

As the customer has already indicated their intent to use Klarna as their payment method, the `presentment()` returns the instruction to show only Klarna and hide other payment methods.

JavaScript

```
{
  "instruction": "SHOW_KLARNA_ONLY",
  "paymentStatus": "REQUIRES_CUSTOMER_ACTION",
  "presentment_options": [
    { "paymentOptionId": "cGF5bW...VudI=", "preselected": true }
  ],
  "payment_descriptors": [
    {
      "paymentOptionId": "cGF5bW...VudI=",
      "content": {
        "nodes": [
          { "name": "PAYMENT_DESCRIPTOR", "value": "Pay in 4" },
          { "name": "PAYMENT_DESCRIPTOR_SUBHEADER", "value": "Pay in 4 interest-free payments of $42.50" },
        ]
      }, {
        "paymentOptionId": "cGF5bWVudC1..NDg4deWs",
        "content": { ... }
      }, ...
    ],
    "impressionUrl": "https://..."
  }
}
```

Select payment method

Pay in 4 Klarna
Pay in 4 interest-free payments of \$42.50

Shop securely with buyer protection. [Learn more](#)

Continue with Klarna

2.7.5.2 When instruction is to preselect one Klarna option

In this example, the `presentment()` returns the instruction to preselect one of the multiple Klarna payment options available to the consumer, and Pay in 4 must be preselected.

JavaScript

```
{
  "instruction": "PRESELECT_KLARNA",
  "paymentStatus": "REQUIRES_CUSTOMER_ACTION",
  "presentmentOptions": [
    { "paymentOptionId": "cGF5bW...deWs", "preselected": false},
    { "paymentOptionId": "cGF5bW...40DI=", "preselected": true},
    { "paymentOptionId": "cGF5bW...deWs", "preselected": false}
  ],
  "paymentDescriptors": [
    {
      "paymentOptionId": "cGF5bWVudC1..40DI=",
      "content": {
        "nodes": [
          { "name": "PAYMENT_DESCRIPTOR", "value": "Pay in 4" },
          { "name": "PAYMENT_DESCRIPTOR_SUBHEADER", "value": "Pay in 4 interest-free payments of $42.50" },
        ]
      }, {
        "paymentOptionId": "cGF5bWVudC1..NDg4deWs",
        "content": { ... }
      }, ...
    ],
    "impressionUrl": "https://..."
  }
}
```

Select payment method

Credit card VISA MasterCard

Pay now Klarna
Pay in full today

Pay in 4 Klarna
Pay in 4 interest-free payments of \$42.50

Shop securely with buyer protection. [Learn more](#)

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

Apple Pay Apple Pay

Continue with Klarna

2.7.5.3 When instruction is to show Klarna

In this example, the `presentment()` returns the instruction to show all Klarna options but not preselect any and respect usual Merchant logic.

JavaScript

```
{
  "instruction": "SHOW_KLARNA",
  "paymentStatus": "REQUIRES_CUSTOMER_ACTION",
  "presentmentOptions": [
    { "paymentOptionId": "cGF5bW..deWs", "preselected": false},
    { "paymentOptionId": "cGF5bW..40DI=", "preselected": false},
    { "paymentOptionId": "cGF5bW..deWs", "preselected": false}
  ],
  "paymentDescriptors": [
    {
      "paymentOptionId": "cGF5bWVudC1..40DI=",
      "content": {
        "nodes": [
          { "name": "PAYMENT_DESCRIPTOR", "value": "Pay in 4" },
          { "name": "PAYMENT_DESCRIPTOR_SUBHEADER", "value": "Pay in 4 interest-free payments of $42.50" },
        ]
      }
    },
    {
      "paymentOptionId": "cGF5bWVudC1..NDg4deWs",
      "content": { ... }
    }, ...
  ], ...
  "impressionUrl": "https://..."
}
```

Select payment method

Credit card VISA

Pay now Klarna
Pay in full today

Pay in 4 Klarna
Pay in 4 interest-free payments of \$42.50

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

Apple Pay

Continue to Order Review

2.7.5.4 When the instruction is to hide Klarna

In this example, the `presentment()` returns the instruction to hide Klarna options when loading the payment selector

JavaScript

```
{
  "instruction": "HIDE_KLARNA",
  "paymentStatus": "REQUIRES_CUSTOMER_ACTION",
  "presentmentOptions": [],
  "paymentDescriptors": [],
  "impressionUrl": "https://..."
}
```

Select payment method

Credit card VISA

Apple Pay

Continue to Order Review

2.7.5.6 Cookbook: Support payment presentment instructions in APIs returning available payment methods

Context: The Partner is creating a checkout experience where they own the Payment Selector, but use the Acquiring Partner's APIs to list available payment methods. The Partner forwards the `interoperability_token` in their requests to the Acquiring Partner's endpoints.

When an Acquiring Partner has APIs that allow Partners to retrieve available payment methods with descriptions (e.g. copy content, icons or legal terms) to build a payment selector, the Acquiring Partner must utilize Klarna's Payment Descriptor API to retrieve up-to-date content and return these in their API - including the presentment instruction.



For example::

```
Java
POST api.acquiring-partner.com/v2/payment_methods
{
  "currency": "USD",
  "amount": 17800,
  "locale": "en-US",
  "payment_method_options": {
    "klarna": {
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
  }
}
```

Acquiring Partners should utilize Klarna Payment Descriptor API behind the scene and use the response to construct their own response. Acquiring Partners can refine the response to their own structure as required.

```
Java
POST api-global.klarna.com/v2/accounts/{account_id}/payment/messaging/payment-presentment

{
  "locale": "en-US",
  "payment_amount": 17800,
  "interoperability": {
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

Response:
{
  "instruction": "PRESELECT_KLARNA",
  "payment_status": "REQUIRES_CUSTOMER_ACTION",
  "presentment_options": [
    { "payment_option_id": "cGF5bW..deWs", "preselected": false},
    { "payment_option_id": "cGF5bW..4ODI=", "preselected": true},
    { "payment_option_id": "cGF5bW..deWs", "preselected": false}
  ],
  "payment_descriptors": [
    {
      "payment_option_id": "cGF5bWVudC1..NDg4ODI=",
      "content": {
        "nodes": [
          { "name": "PAYMENT_DESCRIPTOR", "value": "Pay in 4" },
          { "name": "PAYMENT_DESCRIPTOR_SUBHEADER", "value": "Pay in 4 interest-free payments of $42.50" },
        ]
      }
    },
    {
      "payment_option_id": "cGF5bWVudC1..NDg4deWs",
      "content": { ... }
    },
    ...
  ],
  "impressionUrl": "https://..."
}
```

The response from the Acquiring Partner to the Partner's "payment_methods" request would be constructed like this:

```

Java
{
  "paymentMethods": [
    {
      "name": "VISA",
      "type": "scheme"
    },
    {
      "name": "Pay with Klarna",
      "type": "klarna",
      "presentment_instruction": "PRESELECT_KLARNA"
    },
  ],
}

```

2.7.5.7 Cookbook: Accept a Payment with Klarna requiring partner confirmation

Context: The Partner is expecting to accept a Payment with Klarna using the Acquiring Partner's APIs, but wants to either show a payment selector or run stock checks prior to finalizing the payment.

The Partner uses the Acquiring Partner's API to register a payment and shares the `interoperability_token` in the Klarna specific data points.

```

Java
POST api.acquiring-partner.com/v2/payments
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
  }
}

```

Pass the `interoperability_token` to Klarna when creating a payment request:

```

Java
POST https://api-global.klarna.com/v2/accounts/{account_id}/payment/requests
{
  "currency": "USD",
  "payment_amount": 17800,
  "interoperability": {
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

```

The Acquiring Partner should now launch the purchase flow for the customer, based upon Klarna's response. The Payment Request can be in two states: SUBMITTED or PENDING_CONFIRMATION:

```
JavaScript
// When re-launching the purchase flow is required
{
  "state": "SUBMITTED"
  "state_context": {
    "payment_distribution": {
      "url": "https://pay.klarna.com/1/<>",
    }
  },
  "currency": "USD",
  "payment_amount": 17800
}

// When launching the purchase flow must be skipped
{
  "state": "PENDING_CONFIRMATION"
  "state_context": {
    "payment_confirmation_token": "krn:payment:eu1:confirmation-token:xx"
  },
  "currency": "USD",
  "payment_amount": 17800
}
```

When the `interoperability_token` provided had a payment status `PENDING_PARTNER_CONFIRMATION`, in most cases the Payment Request will automatically transition to `PENDING_CONFIRMATION` and no additional action will be required from the customer to complete the transaction.

2.7.5.8 Cookbook: Accept a Payment with Klarna without further customer interaction

Context: the Partner is expecting the Acquiring Partner to accept a Payment with Klarna using the Acquiring Partner's APIs. They are passing the `interoperability_token` and are expecting to get back an approved payment. Depending on the Acquiring Partner APIs, they may or may not receive a step-up flow.

In this case, the Payment Request with Klarna can be created right away by the Acquiring Partner, passing the `interoperability_token` and asking for the payment to be automatically finalized – meaning that a Payment Transaction should be created immediately:

```
Java
POST /v2/accounts/{account_id}/payment/authorize
{
  "currency": "USD",
  "payment_amount": 17800,
  "require_confirmation": FALSE,
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

⚠ The `require_confirmation` flag can be sent regardless of the presence of the `interoperability_token` and its status. Hence, the Acquiring Partner should not need to read the interoperability status to decide whether the flag should be sent, rather, it's decided on the fact that the Partner must be able to handle an approved payment right after submission.

The Acquiring Partner should now read the status of the response. Klarna should approve the purchase immediately, but could trigger a step-up flow dependent upon the risk evaluation. To enforce the step-up flow, please read [Cookbook: Accept a Payment with Klarna requiring partner confirmation](#).

```
Java
// In case the payment transaction is created immediately:
{
  "payment_status": "PAYMENT_CONFIRMED",
  "payment_transaction": {
    "payment_transaction_id": "krn:payment:eu1:transaction:xxx",
    "currency": "USD",
    "payment_amount": 17800,
  }
}

// In case step-up is required:
{
  "payment_status": "CONSUMER_ACTION_REQUIRED",
  "presentment_instruction": "SHOW_ONLY_KLARNA",
  "payment_request": {
    "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
    "state": "SUBMITTED",
    "state_context": {
      "payment_distribution": {
        "url": "https://pay.klarna.com/l/<>",
      }
    },
    "currency": "USD",
    "payment_amount": 17800
  }
}
```

When an `interoperability_data` provided has a payment status `PENDING_PARTNER_CONFIRMATION`, then in most cases there is no additional action required from the customer to complete the transaction.

2.7.5.9 Cookbook: Create payment token based on `interoperability_token` for recurring payments

Context: The Partner is expecting to enable recurring payments (subscriptions, offline payments) with their Acquiring Partner while they get the consent of the customer from previous interactions with Klarna, such as Sign in with Klarna.

When a Partner would like to set up a tokenized payment (e.g. subscription, recurring payment etc.) with their Acquiring Partner, Partner can forward the `interoperability_token` to Acquiring Partner with intent of tokenization.

An example scenario would be when the Partner has a direct integration with Sign-in with Klarna, which supports payment tokenization. An `interoperability_token` will be generated and shared with Partner when the customer goes through SIWK. Partners will then share the `interoperability_token` with their Acquiring Partners when initiating a payment with intent of tokenization.

```
Java
POST api.acquiring-partner.com/v2/payments
{
  "currency": "USD",
  "amount": 17800,
  "mode": "subscription",
  "payment_method_options": {
    "klarna": {
      "interoperability_data": "...",
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
  }
}
```

Acquiring Partners can then initiate a Payment Request as per standard flows for tokenization with Klarna. Thanks to the `interoperability_token` that was passed, the Payment Request response will contain a `customer_token`. Acquiring Partners can store this in their system and use it for charging the customer with or without customer's presence. The Acquiring Partner returns their own token identifier to Partner for future payments.

```
Java
POST api-global.klarna.com/v2/accounts/{account_id}/payment/authorize
{
  "currency": "USD"
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  },
  "config": {
    "request_customer_token": {
      "scopes": ["payment:customer_not_present"]
    }
  }
}

Response:
{
  "klarna_customer": {
    "customer_token": "krn:partner:eu1:customer-token:xxx"
  }
}
```

Acquiring Partners can use `customer_token` to create future payment requests with Klarna.

```
Java
POST https://api-global.klarna.com/v2/accounts/{account_id}/payment/token/charge
['X-Klarna-Customer-Token']: "krn:partner:us1:live:identity:customer-token:vVQGmY..."

{
  "currency": "USD",
  "payment_amount": 17800,
}
```

2.7.5.10 Cookbook: Retrieve and pass interoperability on post-purchase APIs

Acquiring Partners must accept Klarna's `interoperability_token` and `interoperability_data` on all touchpoints where post-purchase operations are handled. This allows partners to enrich the customer's post-purchase experience and provide better dispute handling. For example:

```
Java
POST api.acquiring-partner.com/v2/payment/xxxxxxxx/capture
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
      "interoperability_data": "...",
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
    }
  }
}
```

The Acquiring Partner then forwards the `interoperability_token` and `interoperability_data` to Klarna via the Payment Transaction API:

```
Java
POST api-global.klarna.com/v2/accounts/{account_id}/payment/transactions/xxx/capture
{
  "capture_amount": 1000,
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

POST api-global.klarna.com/v2/accounts/{account_id}/payment/transactions/xxx/refund
{
  "refund_amount": 1000,
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```



```

PATCH api-global.klarna.com/v2/accounts/{account_id}/payment/transactions/xxx
{
  "supplementary_purchase_data": {
    "purchase_reference": "xxx"
  }
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

POST api-global.klarna.com/v2/accounts/{account_id}/payment/transactions/xxx/authorize
{
  "payment_amount": 2000,
  "interoperability": {
    "interoperability_data": "...",
    "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}

```

2.7.6 Interoperability requirements on Plugins for Platforms

When an Acquiring Partner distributes plugins for Ecommerce Platforms (e.g., WooCommerce, Adobe Commerce) where they own the integration calls between the plugin and their own system, they are required to retrieve the `interoperability_token` and use it, as this cannot be done by the Partner themselves. When receiving it from their plugin, it should be used the same way as if they had received it from the Partner directly.

Requirement	Description
Platform Requirement #1 Acquiring Partners' Platform Plugins should retrieve interoperability data points stored by Klarna Plugins	If the integration is done through a plugin on an e-Commerce Platform (e.g. WooCommerce, Adobe Commerce etc.), the Acquiring Partner plugin must retrieve the interoperability data points from the Klarna Plugin via shared storage in the Platform.
Platform Requirement #2 Acquiring Partners' Platform Plugins should forward interoperability data points to their Payment Platform	Acquiring Partner needs to forward properly the interoperability data points when interacting with their Payment Platform as a Partner would do.
Platform Requirement #3 Acquiring Partner's Plugin retrieves Payment Status and fast tracks payment finalization	Acquiring Partner needs to read Payment Status, if it's pending confirmation then they should try to proceed to the payment with Klarna directly, if not, then proceed to regular checkout.



Requirement	Description
Platform Requirement #4 Acquiring Partners' Platform Plugins should build a compliant payment selector when applicable	When the payment selector is being built by the Acquiring Partner directly in their Platform Plugin, then the usual requirements apply for Presentment Instructions and for sharing the <code>interoperability_token</code> with Klarna's Web SDK.

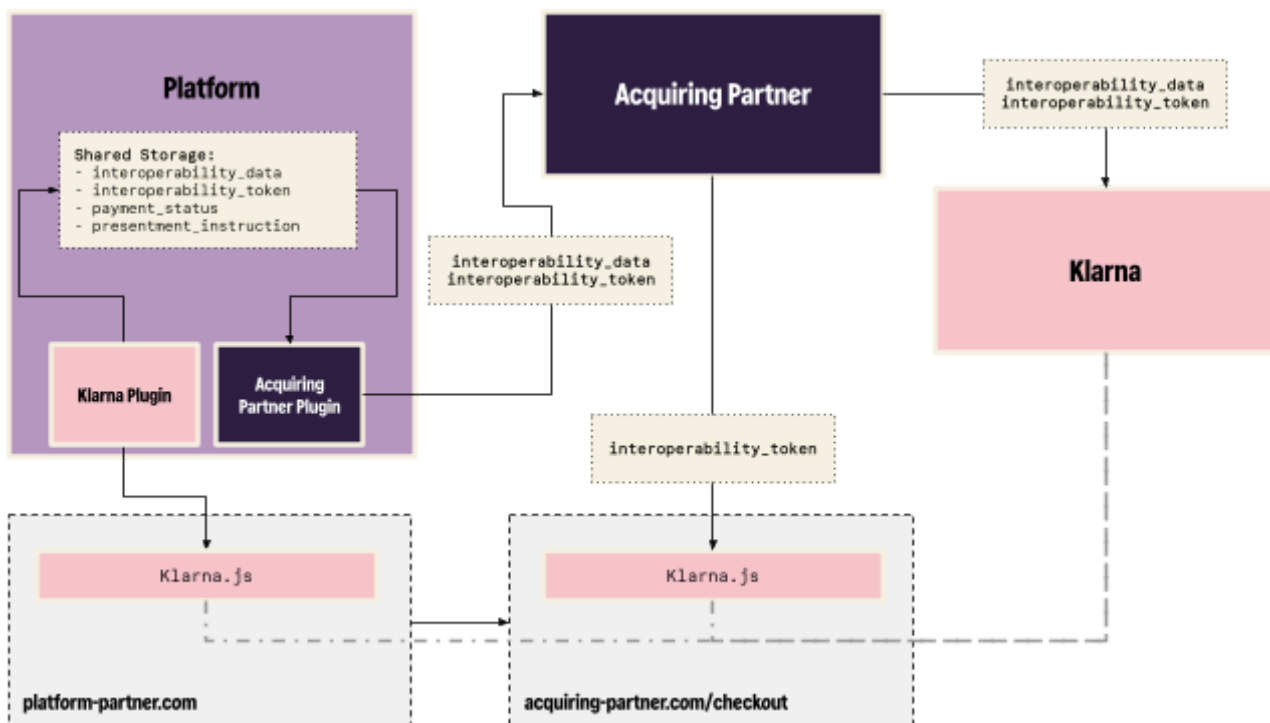
2.7.6.1 How and when to retrieve Interoperability data points

In case the plugin is simply leveraging the Acquiring Partner infrastructure to accept payments, then it will only need to retrieve the interoperability data points before starting the checkout process.

Retrieving interoperability data points will be specific to each ecommerce Platform, but it is usually required to read the session storage of the platform for this customer before handing over to the Acquiring Partner's payment infrastructure.

For Klarna's full product suite to function seamlessly across the customer journey on Platforms, it is required from the Acquiring Partner to retrieve the data points at the last touch point before moving to their own infrastructure. This is usually at the point where the consumer starts the checkout process.

When a Partner utilizes a Platform Plugin maintained by Klarna, the plugin will handle the integration of Klarna's product suite outside of payments and store interoperability data points inside the shared storage. The Acquiring Partner has to then forward them to their own platform.



2.7.6.2 Cookbook: Retrieve interoperability data points by Platforms

As each ecommerce platform is handling checkout sessions differently, here are specifics for each of the platforms.

The data points that will be stored:

Data point	Type	Availability
interoperability_data	String (nullable)	Checkout Sessions, Order data Server-side only.
interoperability_token	String (nullable)	Checkout Sessions, Order data Server-side and client-side.
interoperability_payment_statuses	Enum (nullable)	Checkout Sessions Server-side and client-side.
interoperability_presentment_instruction	Enum (nullable)	Checkout Sessions Server-side and client-side.

Each data point may be independently non-present (null).

2.7.6.2.1 WooCommerce

The interoperability data points will be stored in the session for the WooCommerce customer. This data can be retrieved by reading the WooCommerce session using the same key that we set them with, using the following code:

```
JavaScript
WC()->session->get( 'klarna_interoperability_data' );
WC()->session->get( 'klarna_interoperability_token' );
WC()->session->get( 'klarna_interoperability_payment_status' );
WC()->session->get( 'klarna_interoperability_presentment_instruction' );
```

2.7.6.2.2 Adobe Commerce

The interoperability data points will be stored in the Adobe Commerce checkout session and can be retrieved in the following way:

```
JavaScript

public function getInteroperabilityData() {
    return $this ->
        checkoutSession->getKlarnaOfficialInteroperabilityData();
}

public function getInteroperabilityToken() {
    return $this ->
        checkoutSession->getKlarnaOfficialInteroperabilityToken();
}

public function getInteroperabilityPaymentStatus() {
    return $this ->
```

```

        checkoutSession->getKlarnaOfficialInteroperabilityPaymentStatus();
    }

    public function getInteroperabilityPresentmentInstruction() {
        return $this ->
            checkoutSession->getKlarnaOfficialPresentmentInstruction();
    }

```

2.7.6.2.3 Prestashop

The interoperability data points will be present in session storage and will be accessible (in JavaScript) in the following way:

```

JavaScript
sessionStorage.getItem('klarna_interoperability_data');
sessionStorage.getItem('klarna_interoperability_token');
sessionStorage.getItem('klarna_interoperability_payment_status');
sessionStorage.getItem('klarna_interoperability_presentment_instruction');

```

2.7.6.2.4 Salesforce Commerce Cloud

The interoperability data points will be set in the session's privacy object. The Acquiring Partner can use the below code to retrieve the interoperability data from the SFCC session:

```

JavaScript
var interoperability_data = session.privacy.klarna_interoperability_data
var interoperability_token = session.privacy.klarna_interoperability_token
var payment_status = session.privacy.klarna_interoperability_payment_status
var presentment_instruction =
session.privacy.klarna_interoperability_presentment_instruction

```

2.7.6.2.5 SAP Commerce

In the SAP Commerce plugin, the interoperability data points will be available as a session attribute with their respective identifiers.

On the server side (Java), they can be accessed using SAP Commerce - SessionService as follows, provided the Acquiring Partner plugin code is running the SAP commerce platform context:

```

JavaScript
sessionService.getAttribute("klarna_interoperability_data");
sessionService.getAttribute("klarna_interoperability_token");
sessionService.getAttribute("klarna_interoperability_payment_status");
sessionService.getAttribute("klarna_interoperability_presentment_instruction");

```

On the client side (Javascript), it can be accessed from the html attribute as follows:

```
JavaScript
var interoperability_token = $("#klarna_interoperability_token") .val();
var payment_status = $("#klarna_interoperability_payment_status").val();
var presentment_instruction = $("#klarna_interoperability_presentment_instruction").val();
```

2.7.6.3 Cookbook: Forward interoperability data points to Klarna

Context: A Partner is using Klarna Express Checkout with Klarna's Plugin for a given platform, alongside with the Acquiring Partner's own plugin to accept payments. The Partner enables the Klarna Express Checkout 1-step flow and expects payments to be automatically approved when reaching the payment step.

In a usual integration context, it is on the Partner to ask the Acquiring Partner to register a payment that is ready to be taken with Klarna. In an ecommerce Platform, it is on the Acquiring Partner plugin to trigger requests towards their payment platform, and it is also on them to share the interoperability data points.

When the payment process is initiated by the Acquiring Partner, it is required to retrieve the stored `interoperability_token` and `interoperability_data` from the session context and forward the data to the Acquiring Partner infrastructure when needed.

It is encouraged to store this data for later use - like for displaying Klarna properly in the payment selector:

```
JavaScript
function prepare_klarna_data_points($request) {

    $requests -> payment_options -> set_klarna(
        WC()->session-> get( 'klarna_interoperability_data' ),
        WC()->session-> get( 'klarna_interoperability_token' )
    );

}
```

This would result in the following call to the platform:

```
JavaScript
POST api.acquiring-partner.com/v2/payments
{
  "currency": "USD",
  "amount": 17800,
  "payment_method_options": {
    "klarna": {
      "interoperability_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "interoperability_data": "..."    }
  }
}
```

```
}  
}
```

Data would then be handled as in [Cookbook: Accept interoperability data points from the Partner through Payment & Checkout APIs](#).

2.7.6.4 Cookbook: Fast track payments made through 1-step Klarna Express Checkout

Context: A Partner is using Klarna Express Checkout with Klarna's Plugin for a given platform, alongside the Acquiring Partner's own plugin to accept payments. The Partner enables the Klarna Express Checkout 1-step flow and expects payments to be automatically approved when reaching the payment step.

For payments to be automatically accepted when already approved by Klarna through the Klarna Express Checkout 1-step flow, it is required to read and handle the `payment_status` to correctly define what is the intent of the Partner.

- On `PENDING_PARTNER_CONFIRMATION` payment should be automatically processed with Klarna.
- On `REQUIRES_CUSTOMER_ACTION` the customer should go through the usual payment flow.

```
Unset  
function initiate_payment_flow() {  
  
    $klarna_payment_status = WC()->session->  
        get( 'klarna_interoperability_payment_status' );  
  
    // The payment is ready with Klarna  
    if("PENDING_PARTNER_CONFIRMATION" == $klarna_payment_status) {  
  
        this -> proceedToPaymentWithKlarna(WC()->session);  
  
    }  
  
    // Apply regular logic  
    this -> proceedToPayment(WC()->session);  
  
}
```

The plugin would then re-connect with [Cookbook: Accept a Payment with Klarna without further customer interaction](#).

2.7.6.5 Cookbook: Build a compliant Payment Selector from within the Platform

Context: A Partner is using Sign in with Klarna through Klarna's Plugin for a given platform. They are also using the Acquiring Partner's plugin, which builds directly the Payment Selector from within the code running on the platform.

Here, the platform plugin should follow these same guidelines: [Cookbook: Build a payment selector respecting presentment instructions](#).



2.8 Managing Payment Transactions

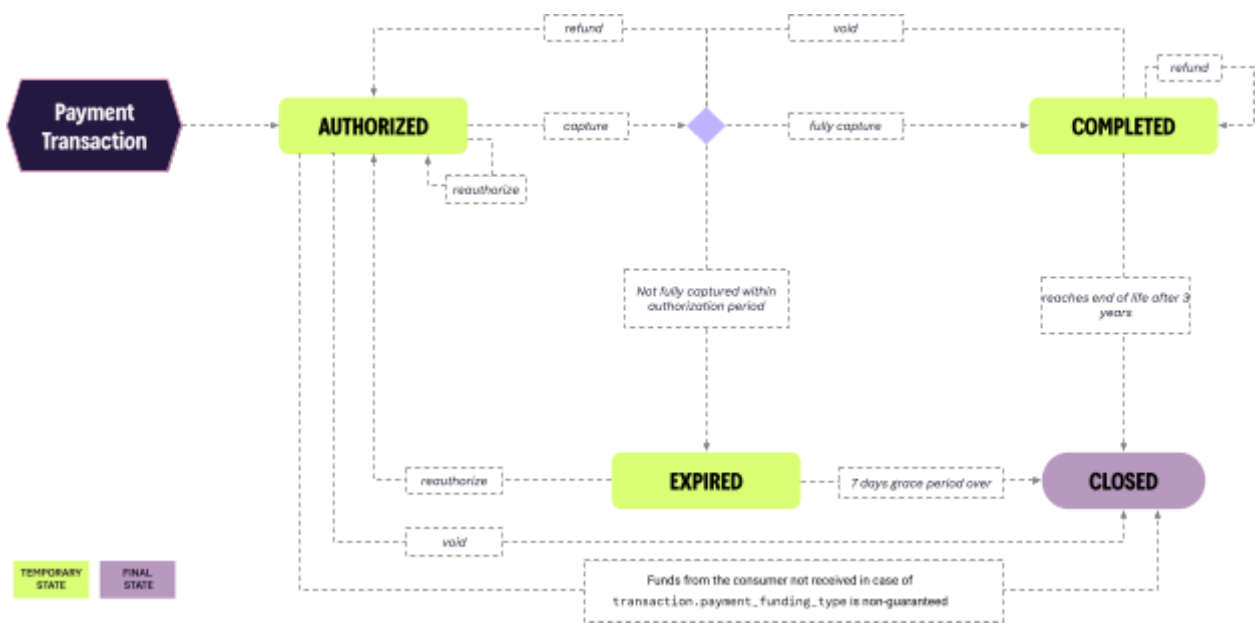
The Payment Transaction API allows Klarna's Partners to track the lifecycle of a payment transaction and perform all post-purchase operations including transaction updates, captures, and refunds.

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.
COMPLETED	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. <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	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. <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).





The lifecycle of a payment transaction involves a sequence of events from authorization through to conclusion. Below is a table describing the events supported by Klarna webhooks for payment transaction state change.

Event name	When
Payment.transaction.state-change.authorized	To track when a payment transaction has an authorized amount remaining and is awaiting further actions like capture, refund, cancel.
Payment.transaction.state-change.completed	To track when a payment transaction has been finalized through funds transfer and no authorized amount remains.
Payment.transaction.state-change.expired	To track when a payment transaction reaches 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.
Payment.transaction.state-change.closed	To track when a payment transaction reaches a definitive conclusion, where no further operations, including refunds, can occur.

The following example reflects the payload structure for Payment.transaction.state-change.authorized:

Example:

```
Unset
{
  "metadata": {
    "event_type": "payment.transaction.state-change.authorized",
    "event_id": "7f1ff389-7792-4cc5-8ec5-cb2ed6e1f19c",
    "event_version": "v2",
    "occurred_at": "2024-01-01T13:00:00Z",
    "correlation_id": "2d1557e8-17c3-466c-924a-bbc3e91c2a02",
```



```

    "subject_account_id": "krn:partner:global:account:live:HGBY07TR",
    "recipient_account_id": "krn:partner:global:account:live:LWT2XJSE",
    "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_transaction_id":
"krn:payment:us1:transaction:4b8b6350-6b72-42c5-b946-f088adcdc216",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "payment_amount": 2100,
    "currency": "USD",
    "state": "AUTHORIZED",
    "state_reason": "AUTHORIZED",
    "remaining_authorization_amount": 2100,
    "customer_id": "krn:kuid:vxsqIkCHf51CWmaZ2beqwz",
    "created_at": "2024-01-01T13:00:00Z",
    "expires_at": "2024-01-29T13:00:00Z"
  }
}

```

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

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 https://api-global.klarna.com/v2/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,
"payment_transaction_reference": "payment-transaction-reference-1234",
"payment_funding": {
  "type": "GUARANTEED",
  "state": "FUNDED"
},
"supplementary_purchase_data": {
  "purchase_reference": "merchant-order-reference-5678",
  "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_identifier": "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"
      }
    }
  ]
},
"payment_captures": [],
"payment_refunds": [],
"payment_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,
"customer_profile": {
  "customer_id": "krn:kuid:vxsqIkChf5lCWmaZ2beqwz",
  "given_name": "Klara",
  "family_name": "Angel",
  "email": "klara.angel@klarna.com",
  "email_verified": true,
  "date_of_birth": "1990-01-01",
  "locale": "en-US"
}

```


```

    },
    "payment_pricing": {
      "applicable_rate": {
        "fixed_fee": {
          "amount": 100,
          "currency": "USD"
        },
        "variable_fee": {
          "percentage": 120
        },
        "unit": "CAPTURE"
      },
      "evaluation_parameters": {
        "price_plan_id":
"krn:partner:global:payment:price-plan:2a3ced5d-270b-319d-7aa4-a4bcf3a2f4b6",
        "program_id":
"krn:partner:global:pricing:payments:program:a5cd46f7e-573d-4b02-bab8-1f1c0e343291",
        "country": "US",
        "merchant_category_code": "5117",
        "channel": "ECOMMERCE",
        "evaluated_at": "2024-01-01T13:00:00Z"
      }
    }
  }
}

```

2.8.1.1.1 Non-guaranteed transactions

Release notes

 Support for non-guaranteed transactions will be added in Release 5.

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 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": {
  "type": "GUARANTEED | NON_GUARANTEED"
  "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": {
      "type": "NON_GUARANTEED"
      "state": "FUNDED"
      "previous_state": "PENDING",
    }
  }
}
```

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

2.8.1.2 Read Payment Chargeback

The operation should be performed to retrieve the details of a payment chargeback. A payment chargeback encapsulates crucial details, such as the chargeback amount and the chargeback reason which explains why Klarna decided to reverse the transaction.

The possible value for chargeback reason are the following:

- **DISPUTE_MERCHANT_LOSS:** The payment dispute has been finalized in favor of the customer, resulting in a payment chargeback being initiated.
- **CUSTOMER_PAYMENT_DEFAULT:** For non-guaranteed offerings, if Klarna is unable to retrieve funds from the customer and if there were any prior captures performed by the partner, Klarna will register a chargeback for the capture total that will be reflected in your settlement balance. To avoid chargebacks, Klarna suggests capturing transactions only when payment is funded by the consumer.

⚠️ Please note that you can also access this information through the read payment transaction endpoint. However, if you are specifically interested in reading the chargeback, you can use this endpoint.

Response example:

```
Unset
{
  "payment_dispute_id":
  "krn:payment:eu1:transaction:f9669608-a24d-43ef-9f34-ec3d785dc1ae:dispute:1",
  "payment_chargeback_id":
  "krn:payment:eu1:transaction:f9669608-a24d-43ef-9f34-ec3d785dc1ae:chargeback:1",
  "chargeback_amount": 5000,
  "chargeback_reason": "DISPUTE_ARBITRATION_LOST",
  "chargeback_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_identifier": "1234567890"
    }
  ]
}
```

Whenever there is a dispute where arbitration rules in favor of the customer, Klarna will trigger a payment transaction chargeback webhook to which you can subscribe to:

Event name	When
payment.transaction.chargeback	To track successful payment chargeback when dispute arbitration is lost or non-guaranteed payment is defaulted.

2.8.1.3 Update Partner references

This operation should be performed when there is a need to change either the `payment_transaction_reference` or the `purchase_reference`. The `purchase_reference` stores the customer-facing transaction identifier, displayed on the Klarna app and other customer touchpoints. 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

```
{
  "supplementary_purchase_data": {
    "purchase_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",
  "payment_transaction_reference": "new-payment-transaction-reference-1234",
  "payment_amount": 2100,
  "currency": "USD",
  "supplementary_purchase_data": {
    "purchase_reference": "new-merchant-order-reference-5678",
    "line_items": [],
    "shipping": []
  },
  "payment_captures": [],
  "payment_refunds": [],
  "payment_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": {
    "customer_id": "krn:kuid:vxsqIkCHf5lCWmaZ2beqwz",
    "given_name": "Klara",
    "family_name": "Angel",
    "email": "klara.angel@klarna.com",
    "email_verified": true,
    "date_of_birth": "1990-01-01",
    "locale": "en-US"
  },
  "payment_funding": {
    "type": "GUARANTEED",
    "state": "FUNDED"
  },
  "payment_pricing": {
    "applicable_rate": {},
    "evaluation_parameters": {}
  }
}
```

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

2.8.1.4 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.

Request example:

```
Unset
{
  "payment_amount": 4200,
  "supplementary_purchase_data": {
    "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_identifier": "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",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "payment_amount": 4200,
    "currency": "USD",
    "supplementary_purchase_data": {
      "purchase_reference": "merchant-order-reference-5678",
      "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_identifier": "1234567890"
        }
      ]
    }
  }
}
```

```

    },
    {
      "name": "Sales tax",
      "quantity": 1,
      "total_amount": 200,
      "total_tax_amount": 200
    }
  ],
  "shipping": []
},
"payment_captures": [],
"payment_refunds": [],
"payment_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_funding": {},
"payment_pricing": {}
}
}

```

For more information on reauthorizing a payment transaction, refer to the [API reference](#).

2.8.1.5 Reauthorize with new shipping details

This operation should be performed when the shipping information is 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.

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.

Request example:

```

Unset
{
  "supplementary_purchase_data": {
    "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",
    "payment_transaction_reference": "payment-transaction-reference-1234",
    "payment_amount": 4200,
    "currency": "USD",
    "supplementary_purchase_data": {
      "purchase_reference": "merchant-order-reference-5678",
      "line_items": [],
      "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"
          }
        }
      ]
    },
    "payment_captures": [],
    "payment_refunds": [],
    "payment_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_funding": {},
    "payment_pricing": {}
  }
}

```

```
}  
}
```

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

Whenever there is an update related to changes such as quantity adjustment or address modification, Klarna will trigger a payment transaction updated webhook to which you subscribe to:

Event name	When
payment.request.updated	To track successful updates when there are changes to order details, such as quantity adjustments or address modifications, after an order is placed.

2.8.1.6 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",  
    "payment_transaction_reference": "payment-transaction-reference-1234",  
    "payment_amount": 4200,  
    "currency": "USD",  
    "supplementary_purchase_data": {  
      "purchase_reference": "merchant-order-reference-5678",  
      "line_items": [],  
      "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"
    }
  ]
},
"payment_captures": [],
"payment_refunds": [],
"payment_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_funding": {},
"payment_pricing": {}
}
}

```

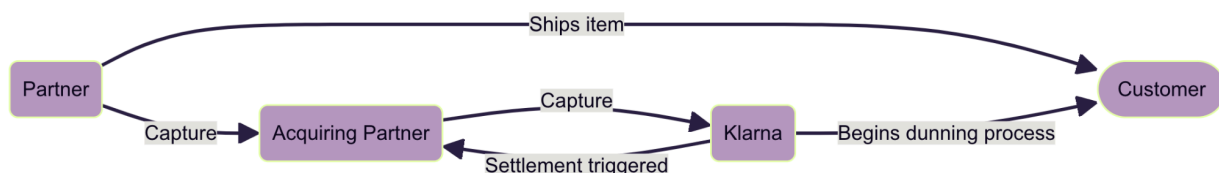
Consult the [API reference](#) for a complete description of the response body parameters.

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 or by viewing 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 **FUNDED**. 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": 2500,
  "payment_capture_reference": "capture-reference-123",
  "supplementary_capture_data": {
    "line_items": [
      {
        "name": "Oversized vintage trabant tee",
        "quantity": 1,
        "total_amount": 2500,
      }
    ]
  }
}
```

```

      "product_url":
"https://merchant.example/product/oversized-vintage-trabant-tee-3344556677",
      "image_url": "https://merchant.example/image/3344556677.png",
      "product_identifier": "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",
  "payment_capture_reference": "capture-reference-123",
  "capture_amount": 2500,
  "captured_at": "2024-01-02T13:00:00Z",
  "supplementary_capture_data": {
    "line_items": [
      {
        "name": "Oversized vintage trabant tee",
        "quantity": 1,
        "total_amount": 2500,
        "product_url":
"https://merchant.example/product/oversized-vintage-trabant-tee-3344556677",
        "image_url": "https://merchant.example/image/3344556677.png",
        "product_identifier": "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 Auto capture

Setting capture to TRUE in the [config object of the payment confirmation 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": "USD",
  "payment_amount": 2000,
  "payment_transaction_reference": "partner-authorization-reference-4567",
  "config": {
    "capture": true
  }
}

```

Response example:

```

Unset
{
  "payment_request_id": "krn:payment:eu1:request:552603c0-fe8b-4ab1-aacb-41d55fafbdb4",
  "state": "CONFIRMED",
  "state_expires_at": "2024-01-01T15:00:00Z",
  "state_context": {
    "payment_transaction": {
      "payment_transaction_id": "krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0",
      "payment_transaction_reference": "partner-transaction-reference-1234",
      "state": "COMPLETED",
      "payment_captures": [
        {
          "payment_capture_id": "krn:payment:eu1:transaction:6debe89e-98c0-486e-b7a5-08a4f6df94b0:capture:1"
        }
      ]
    },
    "payment_funding": { ... },
  }
}

```

```

      "payment_pricing":{ ... }
    }
  },
  "expires_at":"2024-01-02T13:00:00Z",
  "created_at":"2024-01-01T12:00:00Z",
  "updated_at":"2024-01-01T13:00:00Z",
  "currency":"USD",
  "payment_amount":1000,
  "config":{

"redirect_url":"https://partner.example/klarna-redirect?id={klarna.payment_request.id}"
}
}

```

Every successful payment capture operation will trigger a payment transaction captured webhook to which you can subscribe to:

Event name	When
Payment.transaction.captured	To track successful payment capture, whether autocapture or manual, ensuring proper record-keeping.

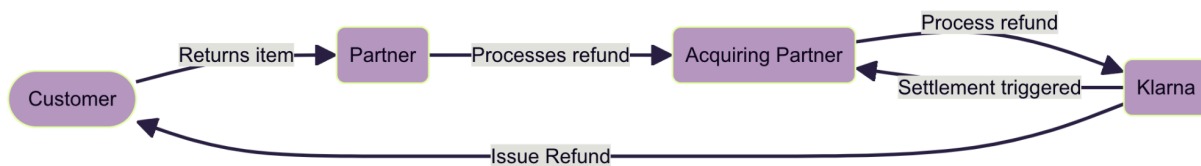
2.8.3 Refund payment transactions and allocation

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

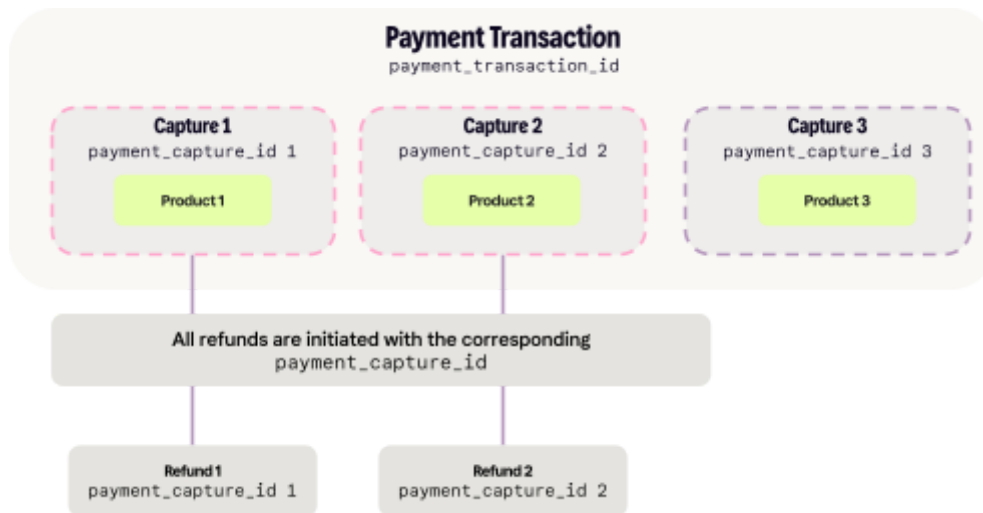
Operation	Requirement	Notes
Refund Payment Capture	Retain the payment_capture_id	This operation simplifies integration for partners that do not have line_items in their integration. Retaining the payment_capture_id and sending it helps Klarna to allocate refund amounts to correct captures.
Refund Payment 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

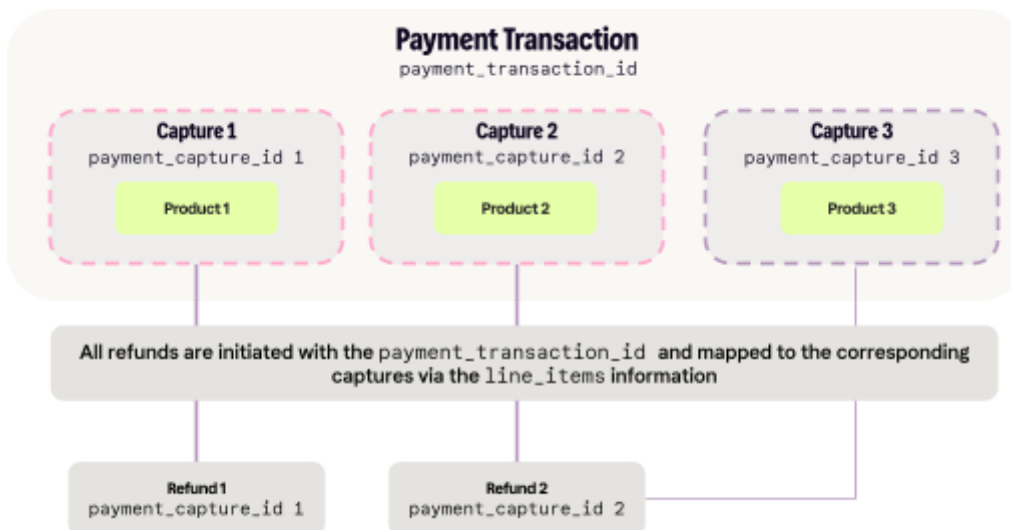
This operation is used to initiate the refund process when customers return items they've purchased. Refunds in this case are allocated to the correct captures through the `payment_capture_id` that is sent to initiate the operation.



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.

Every successful payment refund operation will trigger a payment transaction refund webhook to which you can subscribe to:

Event name	When
<code>Payment.transaction.refunded</code>	To track successful refund initiations ensuring debugging

	any discrepancies.
--	--------------------

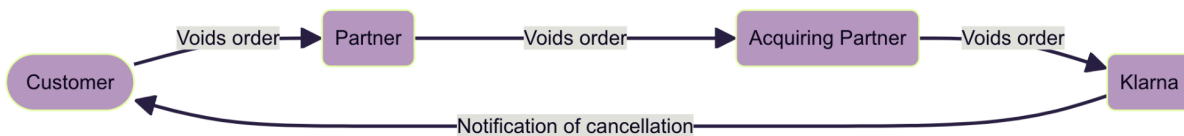
2.8.4 Void payment transaction

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.

2.8.5 Payment Transaction Notifications

Release Notes

17 Payment transaction notifications to be included in subsequent releases.

2.9 Disputes handling

A dispute occurs when a customer questions a payment, often due to reasons like not receiving goods, unauthorized charges, or dissatisfaction with a product. Efficiently managing disputes through Klarna's Dispute API allows Partners to respond in a structured manner, minimize financial impact, and resolve issues promptly.

Acquiring Partners who present disputes within a dashboard are required to integrate Klarna dispute handling, such that the friction of dispute handling for Klarna does not differ from other payment methods. By following these guidelines, Partners can protect their finances and keep strong relationships with customers. Efficiently resolving disputes reduces chargebacks and builds customer loyalty.

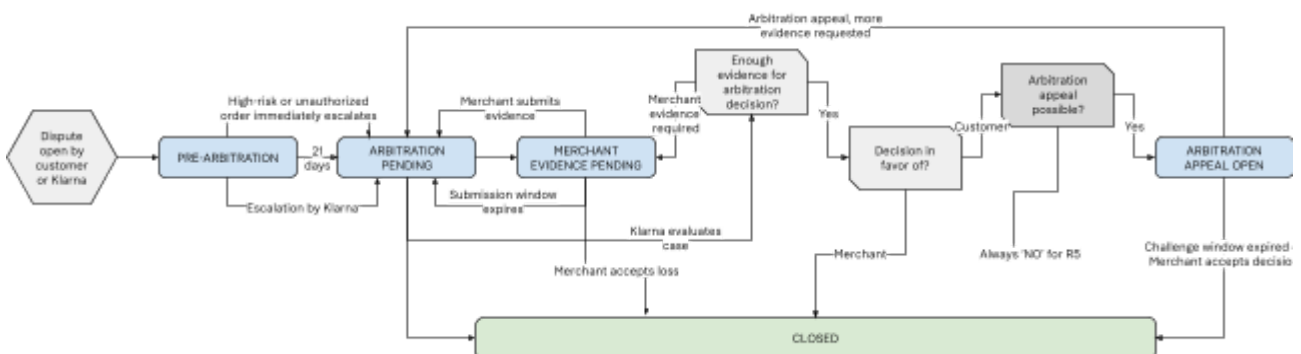
Alternate

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 Detailed Dispute Lifecycle and States

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



Release Note:

 *Arbitration Appeal will be completed in a later release.*

State	Trigger	Webhook	Expected action	Dispute state duration
Pre-arbitration	A customer initiates a dispute.	<code>payment.dispute.state-change.pre-arbitration</code> - Provides the Partner with essential	Klarna alerts the Partner with initial dispute details through a <code>payment.dispute.state-change.pre-arbitration</code> webhook. Partners should review the dispute specifics and prepare any initial	The duration a dispute remains in its initial state varies based on the type of dispute. Immediate Transition to Arbitration Pending: Disputes categorized as High Risk

State	Trigger	Webhook	Expected action	Dispute state duration
		information about the dispute.	response or can accept loss at this stage in order to avoid dispute fees since Klarna will not charge the dispute fee at this stage.	<p>Purchase and Unauthorized Purchase will transition immediately to the next state, which is Arbitration Pending.</p> <p>21-Day Minimum Duration: Disputes related to the following categories will remain in the initial state for a minimum of 21 days before transitioning to Arbitration Pending:</p> <ol style="list-style-type: none"> 1. Return Not Refunded 2. Goods Not Received 3. Faulty Goods 4. Incorrect Transaction 5. Purchase Cancelled <p>Agent Discretion: Klarna has the authority to advance a dispute to the next state prior to the completion of the 21-day period, based on their discretion. These scenarios include, but are not limited to:</p> <ol style="list-style-type: none"> 1. Disputes related to Debt Collection 2. Complaints 3. Claims raised by an external authority (for example ARN in SWE)
Merchant Evidence Pending	Klarna formally requests evidence from the Partner.	<code>payment.dispute.state-change.merchant-evidence-pending - Prompts the Partner to provide requested documentation.</code>	Partners must upload and submit specific evidence via the Dispute API. Attach the attachment ID received after upload in response to the <code>payment.dispute.state-change.merchant-evidence-pending</code> webhook.	Partners are given 14 days to respond to an evidence request. The dispute will remain in this state until evidence is submitted or the window for evidence submission expires.
Closed	The dispute reaches a resolution	<code>payment.dispute.state-change.closed - Confirms that the dispute has been finalized and indicates the outcome.</code>	If the dispute closes in favor of the customer, the Partner may incur a chargeback. If the Partner prevails, no additional action is required.	



2.9.2 Common dispute scenarios

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: Products not received

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 products not received.

Webhook: `payment.dispute.state-change.pre-arbitration`

Expected Action: The Partner should provide tracking details, proof of shipment, and delivery confirmation if available.

Resolution Pathway 1: ShopStore reviews Emily's dispute and checks their shipping records. They confirm that the laptop was shipped but delayed due to an issue with the courier. ShopStore provides tracking information and proof of shipment through Klarna's Dispute API, along with an updated delivery timeline. Emily receives the package and confirms its delivery through Klarna.

Resolution Pathway 2: ShopStore reviews Emily's dispute and finds evidence that the laptop was delivered to the correct address. They submit proof of delivery through Klarna's Dispute API. Klarna reviews the evidence and determines that the package was delivered as promised. The dispute is closed in ShopStore's favor.

Required Evidence: Shipping confirmation, tracking ID, proof of delivery and recipient name for validation.

Example 2: Products faulty

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.

Webhook: `payment.dispute.state-change.pre-arbitration`

Expected Action: Partners should submit photos of the returned item, shipment details, and any customer correspondence.

Resolution Pathway 1: HomeStyle Furniture reviews James's dispute and requests photos of the damaged chair. After verifying the damage through the provided images, they acknowledge the issue. HomeStyle Furniture offers James a replacement chair at no additional cost and ships the new item promptly. HomeStyle provides the evidence of replacement chair by Klarna Disputes API



Resolution Pathway 2: HomeStyle Furniture reviews James's dispute and the provided photos. They find that the chair's condition does not match the description of damage or believe the issue resulted from misuse after delivery. HomeStyle Furniture submits evidence via Klarna's Dispute API. Klarna reviews the evidence, determines that HomeStyle Furniture is not liable, and closes the dispute in their favor.

Required Evidence: Confirmation that changes will be made to order or alternative resolution was arrived at. Transaction IDs and/or numerical discounts in form of attachment is required

Example 3: Capture amount incorrect

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.

Webhook: `payment.dispute.state-change.pre-arbitration`

Resolution Pathway 1: GadgetGrove reviews their payment records and confirms that a duplicate charge occurred due to a system error. They promptly issue a refund for the extra payment through Klarna's Dispute API and provide evidence, such as transaction logs and invoice copies, to support the resolution. Klarna reviews the submission and informs Olivia of the refund.

Resolution Pathway 2: GadgetGrove reviews their payment records and finds no evidence of a duplicate charge. Their investigation shows that the transactions on Olivia's statement correspond to separate orders or activities. GadgetGrove submits evidence via Klarna's Dispute API to demonstrate that the charges were legitimate. Klarna reviews the evidence, determines that the charges are valid, and closes the dispute in GadgetGrove's favor.

Required Evidence: Relevant financial records and invoice copies in form of attachment.

Example 4: Purchase unauthorized

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.

Webhook: `payment.dispute.state-change.pre-arbitration`

Expected Action: GameHub should investigate the purchase and supply any fraud prevention measures taken (e.g., proof of customer ID verification).

Resolution Pathway 1: GameHub investigates the transaction and finds that it was made using stolen card information. They confirm that the purchase was unauthorized and promptly cancel the order. GameHub refunds Michael and provides the required evidence to clarify the situation. Klarna reviews the evidence, confirms the refund, and updates Michael.

Resolution Pathway 2: GameHub investigates the transaction and finds no evidence of fraud. Their review shows that the order was placed from a device and location consistent with previous purchases associated



with Michael's account. They submit evidence via Klarna's Dispute API. Klarna reviews the evidence and determines that the transaction appears legitimate.

Required Evidence: Transaction details, customer verification steps, and any fraud reports.

Example 5: Return not refunded

Scenario: A customer, Sarah, returns an item purchased from FashionFusion, an online apparel store. Despite receiving confirmation from the courier that the return was delivered to FashionFusion's warehouse, Sarah hasn't received her refund after 14 days.

Dispute Trigger: Sarah contacts Klarna to initiate a dispute, claiming "Refund not received."

Webhook: `payment.dispute.state-change.pre-arbitration`

Expected Action: Confirm receipt of returned items; specify condition or reason if not accepted.

Resolution Pathway 1: FashionFusion reviews their return records and confirms receipt of the item. They discover that the refund was delayed due to a processing error. FashionFusion promptly issues the refund to Sarah and notifies her of the resolution through Klarna's Dispute API. Sarah confirms receipt of the refund, and the dispute is resolved and closed.

Resolution Pathway 2: FashionFusion reviews their return records and finds no evidence of receiving the returned item. They submit proof via Klarna's Dispute API, including warehouse logs and courier tracking data, showing that the return was never delivered. Klarna reviews the submitted evidence, determines that FashionFusion is not liable, and closes the dispute in their favor. Sarah is notified of the decision, and the case is resolved.

Required Evidence: Return policy compliance, item photos, or tracking records for returned items, acknowledgement of return.

2.9.2.1 Accepting a 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. **Encourage Partner to communicate with the Customer:** Proactively 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.

2.9.2.2 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 (limited to a maximum of 7.0 MB) 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.

2.9.3 Dispute notifications

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.

2.9.3.1 Step 1: Listen to Dispute Webhooks

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. Dispute webhooks are categorized into two main types: *state changes* and *updates*.

Enablement of webhooks allows for:

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

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



2.9.3.1.1 State Change Webhooks

State change webhooks notify Partners of transitions between dispute states as part of the dispute lifecycle. `payment.dispute.state-change`: Tracks transitions between dispute states below:


Event	Webhook Name	Expected Action
Dispute Created	<code>payment.dispute.state-change.pre-arbitration</code>	Dispute transitions to pre-arbitration as the dispute has been submitted to Klarna. The partner should review dispute details.
Dispute in Arbitration	<code>payment.dispute.state-change.arbitration-pending</code>	Dispute transitions to Arbitration Pending when the normal deadline for resolution has expired. Prepare additional evidence if needed.
Evidence Requested	<code>payment.dispute.state-change.merchant-evidence-pending</code>	Transition to Merchant Evidence Pending when Klarna requests additional information from the Partner. Submit required documentation via API.
Merchant Response	<code>payment.dispute.state-change.arbitration-pending</code>	Transition back to Arbitration Pending when the Partner replies to the request for evidence. Await update from Klarna upon review of evidence.
Dispute Closed	<code>payment.dispute.state-change.closed</code>	Indicates the final resolution of the dispute.. Review final outcome and update records.

2.9.3.1.2 Update Webhooks

Update webhooks provide notifications when dispute details or evidence have been updated, helping partners ensure timely responses and evidence submissions. `payment.dispute.updated` notifies when dispute details or evidence have been updated, helping partners ensure timely responses and evidence submissions. Events are:

Event	Webhook Name	Expected Action
Deadline Extension	<code>payment.dispute.updated.arbitration-deadline-extended</code>	Communicates an extension of the deadline for the Partner to respond.
Dispute Amount Updated	<code>payment.dispute.updated.disputed-amount-updated</code>	Notifies changes to the disputed amount or currency.

2.9.3.2 Step 2: Display Dispute Updates to the Partners

 *Guidance on the presentation of Disputes within your dashboard to come in subsequent releases.*

2.9.3.3 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 final. 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 retrieval of these rates be necessary, you can access them using the [read price plan request](#). To view the broader rates across a Partners price plan, you should use the GET a price plan request. This ensures accurate and up-to-date rate application in all transactions and calculations. The rates provided in the response to the [confirm payment request](#) are only intended for estimation purposes, and should not be stored or used for forecasting or reconciliation.

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 issuing 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.

Microtransaction cap: The microtransaction cap is a limit applied to ensure that the total fee, including both the variable and fixed fee components, does not exceed a predefined percentage of the transaction value. This feature is designed to protect merchant margins on lower-value transactions by capping the maximum fee that can be charged per transaction. The specific cap may vary by country and/or merchant category code (MCC) as outlined in your agreement.

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 response to the [confirm payment request](#) and (once authorized) the [read payment request](#), rate details are disclosed. The `payment_pricing` object shows the details of the costs associated with transaction, with the detailed rate amounts calculated based on these details:

Fee	Definition
<code>applicable_rate.fixed_fee</code>	Fixed fee charged per capture. Fixed fee includes fee amount with a precision of 2 minor units and currency of the fee.
<code>applicable_rate.variable_fee</code>	Percentage fee charged per capture with the precision of 2. E.g: 1.7% is transmitted as 170
<code>evaluation_parameters</code>	Response includes a list of parameters and their values that were used to determine the applicable rate. The evaluation parameters include a <code>price_plan_id</code> , the <code>payment_program_id</code> , <code>country</code> , <code>merchant_category_code</code> , <code>channel</code> and <code>timestamp</code> of evaluation.



evaluation_parameters.price_plan_id	It is an id of the price-plan used to evaluate the rate for the transaction. This id can be used to read the broader rates using Price Plan APIs
evaluation_parameters.payment_program_id	Payment_program_id refers to the payment program used for this transaction.
evaluation_parameters.evaluated_at	The time at which the rate for the transaction was evaluated.

Example


For a captured transaction completed in USD, with a fixed fee of \$1.00 and a variable rate of 1.2%, the below response would be returned. This rate should be considered the “applied rate” and will be adjusted should any incentives or caps be applied.

Response

```
Unset
"payment_pricing": {
  "applicable_rate": {
    "fixed_fee": {
      "amount": 100,
      "currency": "USD"
    },
    "variable_fee": {
      "percentage": 120
    }
  },
  "unit": "CAPTURE",
}
"evaluation_parameters": {
  "price_plan_id":
"krn:partner:global:price-plan:3305b39b-5418-4e46-af22-4c3d5c1e1743",
  "payment_program_id":
"krn:partner:global:pricing:payment:program:a5cd46f7e-573d-4b02-bac8-1f1c0e343291",
  "country": "US",
  "merchant_category_code": "5741",
  "evaluated_at": "2024-01-01T13:00:00Z"
}
}
```

2.10.1.2 Payment programs and identifiers

Release Note

 *The Payment Programs API will not be available until a later release.*

Each payment program is associated with a unique payment_program_id, which should be the primary identifier used by partners to reference and reconcile programs. Program names are provided



as descriptive elements but may vary over time as Klarna expands or updates offerings. This makes program IDs the only stable reference for any program-related processing.

2.10.1.2.1 Retrieving program rate information

For rate details associated with a specific program, partners should use the [Price Plans API](#). This API provides a way to read the price plan. It provides rates based on `mcc`, `payment_program_id`, `country`, `currency`, etc. Partners can query using 1 or more filter parameters to query required rate details.

Confirming Program Rates:

- Query the Price Plans API with the `price_plan_id` and `payment_program_id` to receive the applicable fixed and variable rates for each program.
- Rates retrieved via the Price Plans API should be considered dynamic and should not be stored on your end, as they may vary.

2.10.2 Reconciling Klarna Settlements

Once a transaction is captured, the applicable 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 tax applied to any fees. Details on how to retrieve the information discussed below is outlined in [Settlement report](#).

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 `payment_reference` is included within the metadata in the transfer of funds according to best practices in each market. This id may be used to correlate a given payout to the associated settlement file. Each currency will result in a separate payout with its own `payment_reference`.

2.10.2.2 Relating a transaction to the applied fees

A given transaction can be partially captured or refunded multiple times and fees are applied per capture. As such there may be multiple fees associated with any given transaction.

- The `payment_transaction_id` is provided within the settlement file for all actions regarding a payment transaction.
- The `payment_capture_id/payment_capture_reference` are only associated with a specific capture, and `payment_refund_id/payment_refund_reference` 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 Managing adjustments

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

- **Fees:** The settlement file will include the fees applied to each capture event. These match the effective rates as previously discussed.
 - Tax (VAT/GST) will be applied to the fees if applicable (ie. EU and Australia).



- The fixed and variable rates of the tax will be included in the settlement file in the `tax_amount` field.
- Klarna calculates tax based on the unrounded value of each individual fee on a given capture.
- In the settlement details, `tax_amounts` will be summarized for ease of consumption.
- The fee amount is separated from the tax (`fee_amount + tax_amount = total fee`)
- **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.
- **Other Adjustments:**
 - **Fee Correction:** A correction applied to correct a previously charged fee in cases where an error occurred related to the application of fees.
 - **Chargeback:** Reverses a transaction following a dispute.
 - **Holdback/Release:** Handles holds or releases. Generally used when an account is `UNDER_REVIEW` or has `INSUFFICIENT_BANK_ACCOUNT_DETAILS`.

It is possible that multiple chargebacks may be applied on a single transaction. If this occurs, the transaction details will be included in the settlement as separate lines, ensuring the capture and transaction can be correlated to the chargeback, and the details of the chargeback are also available.

2.10.2.3.1 Carryover debt balances

If a settlement has a negative balance, this negative balance will be reported as `closing_debt_balance` in the report and reflected as an `opening_debt_balance` in the subsequent report.

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:** Buy rates, set by Klarna, are included within the price plans and can be accessed at any time. Sell rates, however, are determined and managed solely by you as the Acquiring Partner.
- **Fee data:** Use the fee 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 rates to Partners

Acquiring Partners must ensure transparent communication regarding the sell rates offered to Partners to maintain clarity and reduce the risk of Partner-related inquiries. Disclosure of the buy rates agreed between Klarna and the Acquiring Partner is prohibited unless explicitly agreed otherwise. Furthermore, sharing or publishing the Klarna Network Price Sheet with merchant Partners is strictly prohibited.

2.10.3 Payment cut-off times

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 normally be available within 24 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 London time UTC in winter, UTC +01:00 in summer
Asia Pacific	00:00 Sydney Time UTC +10:00 in winter, UTC +11 in summer
North America	00:00 New York Time UTC -5 in winter, UTC -4 in summer

2.10.4 Settlement report

Detailed settlement reports are available both via APIs and SFTP. Where settlement occurs directly towards Partners, they may also access Settlement reports via the Klarna Portal.

2.10.4.1 Settlements API

The Settlements API is the preferred method for reconciling settlements, and is ideal for managing and gathering large volumes of settlement reports, offering greater flexibility and scalability than the alternative solutions.

2.10.4.1.1 Steps for Reconciliation:

1. **List Settlements:** Use the List Settlements request to retrieve a list of settlements over a specific period. You can filter results by time frame, currency, or business entity. The response includes sufficient information to match a payment to its associated settlement report.
2. **Retrieve Settlement Report:** Use the [get settlement report](#) to get detailed information about a specific settlement, including the total settlement amount and transaction details.
3. **Extract Key Data:** Review key transaction data such as capture amounts, fees, and VAT.
4. **Transaction-Level Reconciliation:** Use the [list settlement transactions](#) request to retrieve transaction-level details for more granular reconciliation.
5. **Get Payout Details:** Use the GET Payout request to obtain details about a specific payout, including the payout amount, status, and currency.
6. **Compare Totals:** Verify that totals for captures, refunds, fees, and chargebacks match your internal data.
7. **Validate VAT:** Use the tax_amount field to ensure VAT is calculated correctly.

Release notes

 LIST Settlements and GET Payout Details will become available in a subsequent release.

More information on the requests involved in settlements, and the data contained within the detailed settlement file are available in the [API Specifications](#).


2.10.4.1 SFTP settlement reports

The Settlements SFTP is a suitable option for manual handling of settlement reporting as needed, when the Settlements API in combination with the webhooks cannot be used.

2.10.4.1.1 Setting up your SFTP

Using the Klarna credentials API documented in [Account Credentials](#), you can generate SFTP credentials. These credentials, which include a username and password, must be stored securely on your end. As soon as these are created, access to settlement reports will be provided on the Klarna SFTP.

Release notes

 Settlement SFTP will not be available until a future release.

2.10.4.1.2 Retrieving Settlement Files from SFTP

After generating your SFTP credentials as outlined in the previous section, you can begin to retrieve your Settlement Reports from the Klarna SFTP. The reports will be available on the day following the [defined cut-off time](#) for each settlement, and kept for 90 days.

Every Settlement Report in the SFTP follows a specific file-naming convention, using the following format: SettlementID.Timestamp.format

Example

For instance, if your Settlement ID is "876543210", and the report was generated on June 01, 2024, the file name would be "876543210.20240601T000000+0000.csv".

In this example, "876543210" is the Settlement ID specific to the payout. This ID is also included in the header of the Settlement Report. The time stamp element, "20230601T000000+0000", refers to the day and time of report generation, set in UTC at the start of the day.

This systematic naming protocol will aid in easy identification and efficient reconciliation of your settlement reports.

2.10.5.1 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. 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.

Use case	When	Payload	Event name
Payout processed	When the settlement has resulted in a payout, but has not yet been executed. The payout request has been sent from Merchant Ledger.	Settlement Id Settlement Amount Currency Settling business entity id Payout details	settlement.payout.p rocessed



		Link to payout details endpoint	
Payout confirmed	Klarna has successfully executed the payout. This does not guarantee the availability of funds in the receiving bank account.	Settlement Id Settlement Amount Currency Settling business entity id Payout details Link to payout details endpoint	settlement.payout.state-change.confirmed
Payout delayed (new)	When a payout is delayed because of Klarna internal issues.	Settlement Id Settlement Amount Currency Settling business entity id Payout details Reason Link to payout details endpoint	settlement.payout.state-change.delayed
Payout failed	When a payout has failed, either when initiating payout or when the receiving bank rejected it.	Settlement Id Settlement Amount Currency Settling business entity id Payout details Reason Link to payout details endpoint	settlement.payout.state-change.failed
Settlement Report generation delayed	Report generation delayed for internal reasons. This is independent of the actual payout.	Settlement Id Settlement Amount Currency Settling business entity id Payout details	settlement.report.delayed
Settlement Report available on API	When the settlement data from Merchant Ledger has been imported and is available in any format in the API.	Settlement Id Settlement Amount Currency Settling business entity id Payout details Links to reports	settlement.report.created
Settlement Report sent to SFTP	When the settlement reports have been uploaded to the SFTP	Settlement Id Settlement Amount Currency Settling business entity id Payout details	settlement.report.uploaded-to-sftp
New invoice available	When an invoice has been created	Amount Currency Other invoice data?	settlement.invoice.created
Bank account added	When a bank account is added through Merchant Tooling.	Settling business entity Settlement configuration id Bank account details	settlement.configuration.bank-account.added
Bank account removed	When a bank account is removed through Merchant Tooling.	Settling business entity Settlement configuration id Bank account details	settlement.configuration.bank-account.removed



Bank account update	When a bank account is updated through Merchant Tooling.	Settling business entity Settlement configuration id Bank account details (new) Bank account details (old)	settlement.configuration.bank-account.updated
---------------------	--	---	---

More information on settlement webhooks are available in the [API specifications](#).



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



17 *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

17 *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.

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.

 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

 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.

 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.

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.

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.

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 [Management API](#).

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

3.3.1.2.1 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 EUR	NA	NA	Min: 0 EUR Max: 10000 EUR	Min: Max:	Min: 0.1 EUR Max: 14000 EUR
Canada	NA	Min: 35 CAD Max: 1500 CAD	NA	Min: 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	Min: 0 EUR Max: 10000 EUR	NA	Min: 0.1 EUR Max: 14000 EUR

Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
			24 months 360-5000 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 2400-150000 NOK 24 months 3600-150000 NOK	Min: 0 NOK Max: 100000 NOK	NA	NA
Poland	Min: 0 PLN Max: 7000 PLN	Min: 150 PLN Max: 5000 PLN	NA	Min: 0 PLN Max: 20000 PLN	NA	NA
Spain	Min: 0 EUR Max: 500 EUR	Min: 35 EUR Max: 1500 EUR	NA	Min: 0 EUR Max: 4000 EUR	NA	NA
Sweden	Min: 1 SEK Max: 150000 SEK	Min: 300 SEK Max: 849.99 SEK *These thresholds may vary depending on the agreement	6 months 250-150000 SEK 12 months 1200-150000 SEK 18 months 2400-150000 SEK 24 months 3600-150000 SEK	Min: 0 SEK Max: 100000 SEK	Min: 0 SEK Max: 50000 SEK	Min: 1 SEK Max: 150000 SEK
Switzerland	Min: 1 CHF	NA	NA	Min: 0 CHF	NA	Min: 0.1 CHF



Market	Pay Later 30 days	Pay in 3/4	Term loan	Pay by Card	Direct Debit	Direct Bank Transfer
	Max: 2500 CHF			Max: 10000 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
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



0

3.4 Create public documentation for your Partners

This guide outlines the steps and best practices for documenting Klarna's Payment Services and shopping solutions for your Partners. It explains Klarna's customer journey, the benefits for Partners and customers, and the integration process. Whether you're new to Klarna or updating existing documentation, this guide offers clear, actionable steps to help you communicate Klarna's services effectively.

Important: Before releasing your documentation publicly, it must go through a thorough review to ensure accuracy and alignment with Klarna's standards. See [Step 5](#) for details.

3.4.1 Step 1: Introduce your partners to Klarna

When introducing Klarna, provide a clear guide that Partners can use to explain Klarna's payment services and shopping solutions to their customers. Show how Partners can integrate Klarna into their apps and physical locations and how Klarna helps expand reach and increase conversions through a unified payment solution.

3.4.1.1 Overview of Klarna

Start with a succinct overview of Klarna and its benefits for retailers and customers. Use Klarna's pre-approved messaging below to maintain clarity and prevent misinterpretation of our vision, mission, products, and terms.

Pre-approved message: Klarna overview

Since 2005, Klarna has been on a mission to smooth commerce. As an AI-powered global payments network and shopping assistant, Klarna is trusted by 85 million global active shoppers and over 575,000 retailers, offering an ecosystem of innovative payment solutions and smart shopping solutions. Revolutionizing how people shop and pay both online and in-store, Klarna is committed to providing a seamless and secure shopping experience that saves customers time, money, and helps them stay in control of their finances

To find the latest metrics about Klarna, visit the [Klarna.com/business](https://klarna.com/business) homepage for up-to-date information.

3.4.1.2 Key benefits for Partners

When promoting Klarna, focus on the value Klarna brings to Partners and their customers. Klarna's payment solutions enhance the shopping experience and support business growth. Use the following pre-approved messaging to highlight how Klarna increases conversion rates, builds customer loyalty, and drives repeat purchases.

- ✓ Maximize sales
- ✓ Save costs
- ✓ Reach new customers



Pre-approved messages: key benefits for Partners

- **Drive conversion:** by using Klarna's range of customer-friendly and flexible payment options and smart shopping solutions. Partners on average see **20% higher conversion** with Klarna.
- **Increase repeat business:** Klarna's unmatched customer experience promotes loyalty and drives a **45% increase in purchase frequency**.
- **Acquire new customers:** Partners get access to Klarna's vast customer network of over **85 million active shoppers**, lowering acquisition costs and enhancing lifetime value.
- **Best-in-class integration with a single API:** that allows for one-time, auto-updated, and cross-country integrations
- **Seller protection:** Klarna employs advanced security measures and fraud detection to protect transactions, building trust with customers, while assuming credit and fraud risk.
- **Boosted brand visibility,** Klarna enhances a merchant's visibility and reputation, associating them with a trusted and innovative payment solution.
- Partners will always be **paid in full and upfront**, while your customers pay at a later date. Klarna supports high-value orders, and everyday purchases.
- Klarna is a **licensed bank** with regulated credit offerings.

3.4.1.3 Key benefits for customers

When introducing Klarna to your Partners, encourage them to emphasize the convenience and ease of use that Klarna's payment options provide to their customers. Focus on how Klarna enhances the shopping experience with flexible payment plans, transparent terms, and a user-friendly process. Use pre-approved messaging to help your Partners communicate these benefits to their end customers.

- ✓ Save time
- ✓ Save money
- ✓ Stay in control of your finances

Pre-approved messages: key benefits for customers

- **Flexibility:** Klarna offers **flexible and user-friendly payment solutions** designed to support customers' buying preferences. Its secure and flexible payment options allow customers to shop with confidence and control in-store and online. With Klarna, customers can pay upfront in full or decide to pay over time, improving their purchasing power and financial management. Klarna payments offer a variety of payment options tailored to fit different shopping needs and preferences.
- **Easy to use:** a straightforward integration and user-friendly interface of Klarna at checkout, which enhances the customer's experience by providing a seamless transaction process.
- **No extra costs:** Klarna offers interest-free payment options which do not incur additional charges if the customer pays on time.
- **Buyer protection:** Customers can shop confidently, knowing they're covered by Klarna's buyer and fraud protection.

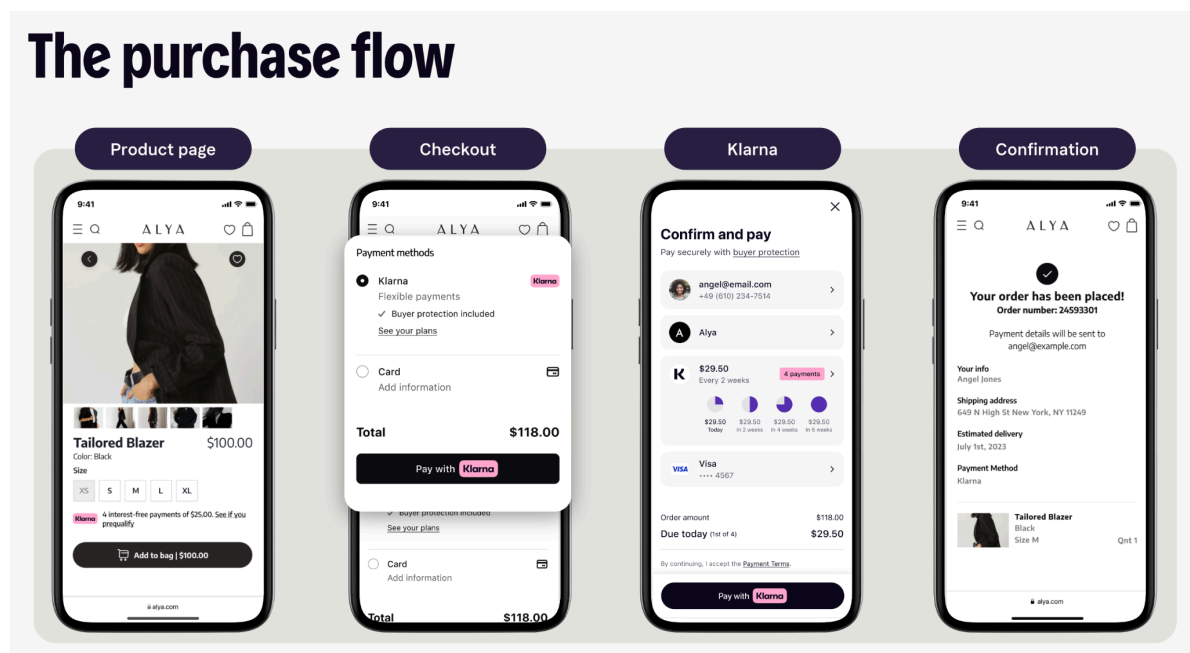


Encourage your Partners to use this messaging to clearly articulate the value Klarna brings to their end customers, enhancing customer satisfaction and driving conversions.

3.4.1.4 Klarna's customer journey

It's important to educate your Partners with a clear overview of the steps customers will take when using Klarna. This ensures Partners understand the flexible and simple customer journey during Klarna's checkout process. Here's an outline of the steps customers will follow when using Klarna:

1. **Klarna pre-purchase flow:** Address the needs and requests of customers early in the shopping journey via Klarna's conversion boosters On-site messaging, Klarna Express Checkout and Sign-in with Klarna.
2. **Select Klarna at checkout:** While shopping online, customers choose Klarna as their preferred payment option at the checkout.
3. **Choose a payment method:** Klarna offers several flexible payment options.
4. **Complete the purchase:** Illustrate the simplicity of completing a purchase with Klarna, emphasizing minimal input requirements by showing how customers enter the required details and confirm their order. Klarna may run a soft credit check for certain payment plans.
5. **Receive Goods and Invoice:** Explain how Klarna provides support once the order is confirmed. Customers receive their purchased items along with an invoice, depending on the payment method chosen.
6. **Manage payments:** Show how customers can track their purchases, review payment installment plans, and make payments accordingly through the Klarna app or website.



3.4.2 Step 2: Provide detailed context regarding your Klarna integration

After introducing your Partners to Klarna, provide detailed guidance on how to integrate Klarna into their online stores. A complete technical integration guide includes several key elements, such as availability, fulfilling Partner requirements, addressing edge cases, and managing more complex solutions.

3.4.2.1 Klarna's availability via your services

It's essential to clearly communicate the availability of Klarna's Payment Services and shopping solutions through your platform. This step involves detailing the specific aspects of service availability,



including geographical reach, technical requirements, and any variations in service options based on Partner or consumer locations.

Keep it simple, leveraging product matrices or interactive pages to not overwhelm Partners. Outline the key elements:

- **Geographical coverage:** Provide a comprehensive list of Acquiring and Issuing Markets where Klarna's services are available through your platform.
- **Product coverage:** Outline any requirements for providing Klarna inherent to your integration, including specific platform or version compatibility, or any other system requirements

3.4.2.2 Integration methods

Explain the different integration methods available for Klarna, such as:

- **API Integration:** For Partners with custom-built e-commerce platforms, provide guidelines on integrating Klarna directly through your API. As this generally affords the Partner more control over presentation, it's especially important to educate these Partners on presentation of Klarna and Klarna branding.
- **Platform Plugins:** Detail the availability of any pre-built plugins for popular e-commerce platforms (e.g., Shopify, WooCommerce, Magento) that simplify the integration process. Ensure Klarna is presented according to best practices through these plugins.
- **Mobile SDK:** If Klarna's Mobile SDK is already built into your mobile solution, let your partners know. If it's not, explain the requirement for them to integrate Klarna's Mobile SDK. Provide clear steps to ensure a successful integration, following Klarna's best practices.
- **Physical Stores:** Provide detailed guidance for Partners on how to enable and implement Klarna in your in-store solutions within their locations.
- **Other paths to enabling Klarna:** Detail your other products, components, SDKs, and payment method integrations to ensure you've clearly explained how to enable Klarna for all partners.

Provide Partners with a link to Klarna-owned platform content at [Klarna Docs - Platform integrations](#).

3.4.2.3 Step-by-Step integration process

Provide a comprehensive, step-by-step guide for integrating Klarna:

1. **Create a Klarna Account:** Direct your Partners through integrating Klarna by detailing the steps within your ecosystem, including how your resources—such as brand information and customer support channels—will be utilized by Klarna.
2. **Access Klarna's Partner Portal:** Guide them on how to log in to the Klarna's Partner Portal and access necessary integration tools and documentation. More information in [Grant access to Klarna's Partner Portal](#).
3. **Configure Integration Settings:** Explain in the context of your integration settings, including API keys, branding options, conversation boosters, disputes, settlements and more.
4. **Install and Configure Plugins:** If applicable, provide instructions for installing and configuring Klarna plugins on their e-commerce platform.
5. **Testing the Integration:** Detail how to conduct sandbox testing to ensure the integration works correctly before going live. Sample test cases are available under [Test Cases](#).
6. **Go-Live:** Provide steps for transitioning from the sandbox environment to the live environment.
7. **Post go-live:** Clarify how to get support and further optimize Partner integrations.

Ensure any technical requirements specific to Klarna within your integration are included within your broader technical documentation. Especially important is listing supported integrations, linking to



relevant API documentation, and providing guidance to ensure Partners provide a secure and safe solution to their customers.

3.4.2.4 Customization and branding

Explain how Partners can optimize and customize the Klarna customer experience, matching their own brand. Access Klarna's [marketing tools](#), which includes a variety of helpful resources and assets to enhance Partner's integration. In the marketing tools, you'll find:

- **[Logo Lock-ups and Klarna Badges](#)**: downloadable assets for use in website banners, badges, and branding.
- **[Pre-approved Klarna Messaging](#)**: ready-made informational and promotional content that Partners can feature on their websites or in-store.
- **[Social Media Kits](#)**: ready-to-use materials for quick and easy promotion of Klarna integration on social media platforms.

Additionally, Acquiring Partners should provide the following, but not limited, marketing materials to Partners:

- **Partner Assets**: detailed guidance on the information required to display their brand within the Klarna ecosystem and instructions on how they can update these assets through their integration.
- **Email Templates**: pre-designed email templates for announcing Klarna to existing customers, making communication simple and consistent.

3.4.2.5 Presenting Klarna within your dashboard

As mentioned in the [introduction](#) to the Integration Guidelines, Klarna offers several non-payment, conversion boosting solutions to Partners, including:

- Express checkout
- On-site messaging
- Sign in with Klarna

To enable these features, Partners must have access to the Partner portal. Technical information on how to implement this is outlined in the [Interoperability](#) section. From a user experience perspective, approved entry points must:

- Include a prominent promotional banner highlighting the value propositions of conversion boosters and including up-to-date Klarna branding.
- Feature a clear call to action for redirection to the Partner portal.

Branding assets and materials are available within [Klarna's Brand Guidelines](#). Where possible, restrict the promotional banner to Partners who have previously completed a transaction with Klarna. The presentation of the entry point must be agreed upon and signed off by Klarna to ensure a mutually advantageous customer experience within your portal.

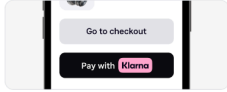
Example: Promotional banner



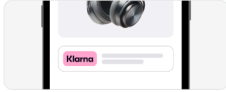
Maximize sales with Klarna

Add these conversion boosters for higher conversion and a better customer experience.

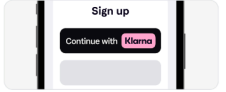
[Get started in Klarna Portal →](#)



Express checkout
Offer a 5x faster check-out process that will lower the threshold for shoppers to complete a purchase.



On-site messaging
Add personalized messaging throughout the shopper journey for higher conversion rates and increased spend.



Sign-in with Klarna
Let customers connect their accounts with pre-set payments and preferences to shop anywhere, improve personalized shopping experiences.

Great features found in the above example:

- The Klarna pink badge is prominently displayed on the banner.
- A brief description of Klarna is included.
- Trackable Call-to-Action (CTA) links directly to the Klarna portal or another relevant Klarna-hosted page.
- All banners have been submitted to Klarna for approval prior to publication.

3.4.2.6 Shopping Session API

3.4.2.67 FAQ compilation

Compile a list of frequently asked questions that Partners might receive from their customers regarding Klarna, and provide clear, concise answers. This resource will help Partners confidently address consumer inquiries and reduce hesitation in the checkout process.

By providing a detailed and Partner-focused introduction to Klarna, you empower them to effectively communicate the advantages to their end consumers, fostering a better understanding and increasing the likelihood of adoption. This approach not only enhances the customer's experience but also supports the Partner's business growth through improved customer satisfaction and sales.

Example: Frequently asked Klarna questions

- **Which business categories can use Klarna payment solutions?** All verticals can use Klarna unless they are prohibited (listed in [Klarna's ethical guidelines](#)). You can read more about the restricted and prohibited businesses [here](#) and if you're unsure, reach out to your Acquiring Partner support.
- **Where are Klarna payments available?** [Here](#) is a list of all countries where Klarna is currently available. Learn more about [Klarna payment offering](#).
- **How can a Partner manage disputes?** The quickest way is for the Partner to contact the customer directly. If unresolved, Klarna can step in by requesting details like return confirmation, tracking ID, and shipping date to assess the dispute. Partners can manage disputes via **[ACQUIRING PARTNER TO ADD: describe where Partners can manage their disputes and receive support if needed]**.
- **Will Klarna increase my Partner's returns?** We encourage Partners to focus on the added benefits Klarna provides, such as improved cash flow, new customers, higher order volumes, and increased average order values, rather than just the return rate. However, partial returns may increase as customers use Klarna to explore their purchase options.
- **When does a Partner get paid?** After the Partner captures (ships) an order, Klarna initiates the payout to the Acquiring Partner, minus any fees, returns, or other charges. The exact payout

date depends on the settlement schedule agreed upon between the Acquiring Partner and the Partner. For more details, reach out to your Acquiring Partner support.

- **Why integrate with Klarna through an Acquiring Partner?** Partners can use their existing integration for quick access to Klarna's payment methods and services, without needing a separate agreement. They also benefit from receiving settlements directly from the Acquiring Partner. With seamless integration, both Partners and customers enjoy the same Klarna experience as a direct integration.

3.4.3 Step 3: Promote Klarna effectively.

3.4.3.1 Provide educational materials for Partners

Supply Partners with educational materials such as tutorials, infographics, or videos that they can use to educate their customers about using Klarna. These resources should be clear, engaging, and easy to distribute, ideally suited for integration into various consumer touchpoints such as product pages, FAQs, or checkout sections of your documentation.

Include testimonials and case studies from other Partners who have successfully integrated Klarna. These real-world examples can help build credibility and illustrate the potential business benefits, such as increased conversion rates and average order values.

Encourage your Partners to adopt Klarna by presenting a clear call to action. Clearly link to resources where they can sign up for or request more information about the [Klarna integration](#).

3.4.3.2 Presentation of Klarna in Partner ecosystem

More information on how to present Klarna to end customers is available within the appropriate checkout sections, however it is required that you drive Partners to present Klarna in line with the steps outlined below.

1. **Educate:** drive Partner to inform customers of the availability of Klarna with informational resources, and ensure Klarna's brand and logo are present wherever payment is discussed or payment information is displayed. Linking out to Klarna-managed resources like the below may be helpful in minimizing maintenance:
 - a. [Klarna Docs - Payment methods availability](#)
 - b. [Klarna Docs - Brand guidelines](#)
 - c. [Klarna Docs - User experience](#)
2. **Klarna conversion boosters :** Partners should leverage Klarna's conversion boosters for online purchases only wherever equivalent customer flows are available. Provide a summary of the benefit of these features and link out to the resources provided below.

Example explanation:

Complete your integration in the Klarna Portal and instantly benefit from the power of Klarna's conversion boosters that are designed to drive sales to your store and level up your customers' shopping experiences.

- a. **Express checkout:** boost conversion with a one-click checkout that offers a smooth and consistent experience to customers, pre-filling details they already set.
- b. **Sign-in with Klarna:** early in the checkout flow, encourage your Partners to enable a faster, easier checkout experience for their customers, leveraging Klarna's vast insight into customer preferences.



- c. **Klarna's On-site messaging:** presents clear information on product pages, at checkout, and elsewhere to ensure customers are aware of their payment options.
 - d. **Presentation in checkout:** where possible, drive the adoption of dynamic presentation of Klarna as outlined in the [checkout guidelines](#) for online purchases only. This ensures compliance and rapid market expansion without additional development, and allows customers to be clearly presented with their options in checkout.
 - e. If Sign-in with Klarna or Express checkout were leveraged during the checkout flow, Partners must leverage the data provided by Klarna to minimize friction and preselect Klarna as these customers have already indicated their preference.
3. **Klarna's media offering:** a performance-based media mix that puts Partners in front of customers at the point of purchase on various Klarna channels, leveraging rich profile information for better attribution. Read more about Klarna's marketing solutions [here](#).
 - a. Ads: Drive better engagement with highly-relevant, targeted ads.
 - b. Search: Reach customers actively searching for retailer's products.
 - c. Affiliate: Deliver higher conversion rates and AOV with Klarna's affiliate partnership.
 4. **Complete the purchase:** educate Partners on when goods should be captured or adjusted. Capture should be triggered when the order is being fulfilled, ensuring the best possible customer experience and fulfilling requirements outlined in Klarna Network Rules. If goods are shipped, encourage the inclusion of tracking data to manage customer expectations and improve satisfaction.
 5. **Post purchase:** drive Partners to act promptly to action refunds or disputes to maximize customer satisfaction.

3.4.4 Step 4: Enable interoperability

To ensure a seamless experience for integrated Partners, it is essential that interoperability is documented in a clear and consistent manner. This involves providing Partners with the necessary tools and information to leverage Klarna's services effectively through your platform.

3.4.4.1 Ensuring Interoperability Availability

It's a must to ensure that interoperability features are accessible to all integrated Partners. This includes

- Enabling the correct fields to be sent to Klarna through your platform and documenting the process thoroughly for Partners.

Proper documentation ensures that Partners understand how to use Klarna's functionality in a way that aligns with both Klarna and your requirements. While key fields and requirements of interoperability are outlined in the [Interoperability](#) section, the method of exposing these tools to your Partners will be unique to your integration. Ensure you provide clear mapping and documentation around these parameters and methods so Partners can fully exploit the benefits of interoperability.

3.4.4.2 Supplementary Purchase Data

Include a section in your documentation explaining the value of Supplementary Purchase Data (SPD). Drive Partners who operate in high-risk verticals to Klarna's resources on leveraging SPD to improve conversion and reduce fraud risks.

SPD refers to additional data which can be provided to Klarna either directly from Partners or through their existing Acquiring Partner relationship to ensure a fuller understanding of the transaction. In cases



where the additional information may be presented via your existing integration, this should be provisioned to Klarna directly.

Educate Partners on the benefits of providing SPD. More detailed information about SPD—including requirements, examples, and technical specifications—can be found in the [Interoperability](#) section. This resource outlines how to structure and transmit the necessary data for smooth interoperability, helping Partners leverage Klarna’s payment solutions effectively.

3.4.5 Step 5: Review your integration with Klarna

The material you create must be reviewed by Klarna. Please reach out to your dedicated Klarna team for assistance with the review process.

By offering detailed support resources and comprehensive training materials, you empower your Partners to implement Klarna effectively. This not only enhances their store's payment flexibility but also contributes to a better shopping experience for their customers. With the right support and resources, Partners can confidently integrate Klarna, leading to improved customer satisfaction and business growth.



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 and dynamically retrieve 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 Ensure 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.mjs 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 [Klarna.mjs](#).

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 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):
 - /v2/accounts/{account_id}/payment/requests
 - HTTP header:
 - Authorization: Basic <api_key>

Release notes

17 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):
 - /v2/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.
 - o Example command:

Unset

```
openssl ecparam -genkey -name prime256v1 -out my-key-file.pem
```

3. Create a Certificate Signing Request (CSR):
 - o The Common Name (CN) in the CSR should be the last part of your account ID.
 - o For example, if your account ID is `krn:partner:global:account:live:LYABCDEI`, the CN should be `LYABCDEI`.
 - o 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 together. 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. After the certificate has been returned, the APIs will return an response header including the status of the MTLs verification. The header returned is "Klarna-Mtls-Verification-Status" and the value is one of "NOT_PRESENT", "VALID" and "INVALID" so that the configuration can be verified beforehand.
6. When the integration has been setup and requests returns "VALID" status contact the Klarna account team and communicate from which time mTLS should be enforced
7. At the activation time the mTLS client certificate will be verified on all API-requests and the status header will stop being returned.

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:

```
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

Integration tagging is a crucial mechanism for improving the integration experience by providing detailed and consistent data to support customer experience, performance tracking, and incident resolution. All operations and interactions with Klarna should be tagged with associated integration partners to ensure granular monitoring and performance. To provide this level of detail, Klarna requires that partners provide all available information with each interaction.


Release notes

 17 Integration tagging is currently under development and is subject to change. It will become available in future releases.

This section provides in-depth instructions and examples for partners on how to implement integration tagging to ensure effective data provision with Klarna.

4.5.2.2 Naming standardization

To enable Acquiring Partners to provide all integrated partners without additional validation, and avoid unnecessary rejections of requests when a new partnership is provided, the naming parameters in Klarna's tagging solution are unvalidated strings, not enums. As such it is required that naming conventions are followed whenever a new application name is provided.

 *Note: In cases where Klarna recognizes a partnership under a different name, Klarna agents may reach out to request the value provided be adjusted to remain consistent across multiple integrators*

4.5.2.2.1 Conventions when providing integrator names

To maximize the possibility of alignment in naming conventions without maintaining a list of all possible modules, Please ensure that your *_name parameters follow the below naming conventions:

- Base any name used off of the **commonly used naming convention** for the application - what they call themselves externally towards customers.
- Use **PascalCase** where multiple words are involved in an application name.



- **Standardize language-specific characters** and convert characters with diacritics or accents to their base form for consistent handling. (e.g. "Rénault" becomes "Renault")
- **Strip out spaces and punctuation** including [-!"#\$%&'*+.,/;<=>?@[\\]^_{}~\|s]
- **Separate the company name from its legal form** (e.g. "Klarna Inc." and "Klarna AB" both become "Klarna")
- **Do not include regions or countries** within an application name where it does not express fundamental differences in the performance of the product. (e.g. "Klarna US" and "Klarna SE" both become "Klarna")

4.5.2.2.3 Predefined platforms

Below is a list of key integrators and originators with predefined strings. All tagging requests involving these partners must match their *_name parameters to the strings below. Contact Klarna Solutions Engineers if your integrator or originator is not listed.

Platform Name	Predefined string
24Nettbutikk	24Nettbutikk
4webs	4Webs
Abicart AB	AbicartAb
ACI Worldwide	AciWorldwide
Addis	Addis
Adobe Commerce (Magento)	AdobeCommerce
Adyen	Adyen
Aera Payment & Identification	AeraPaymentIdentification
Airwallex	Airwallex
Alcalink	Alcalink
Alipay	Alipay
Alphabank	Alphabank
Altapay	Altapay
Antom	Antom
Apexx	Apexx
Aptos	Aptos
AQS	Aqs
Artdinamica	Artdinamica
Askås I & R	AskasIR
Aurus	Aurus
AutoPay	Autopay
AvantiCart	Avanticart
Avensia	Avensia
Axerve	Axerve

Platform Name	Predefined string
Moneris	Moneris
Monext	Monext
Multisafepay	Multisafepay
myPOS	Mypos
Mystore	Mystore
NBG	Nbg
NCR	Ncr
Netgiganten.dk	NetgigantenDk
Nethit Systems	NethitSystems
Netopia	Netopia
Netsite	Netsite
Nexi Group (Nets, Concardis, Computop, Paytrail)	NexiGroup
NHN Commerce	NhnCommerce
NHN KCP	NhnKcp
Nice Payments	NicePayments
Norce (Jetshop/Storm)	Norce
Nordicway	Nordicway
Novalnet	Novalnet
Nuvei	Nuvei
OC Payment	OcPayment
Oceanpayment	Oceanpayment
Odero (TOKEN)	Odero
Ohdigital	Ohdigital
Okisam	Okisam



Platform Name	Predefined string
Ayla Diseño y Tecnología	AylaDisenoYTecnologia
Bank of America	BankOfAmerica
Banorte	Banorte
Banque Populaire	BanquePopulaire
BBVA	Bbva
Big Commerce	BigCommerce
Billwerk+	Billwerk
BlackPepper	Blackpepper
BlueSnap	Bluesnap
Blugento	Blugento
BNPP	Bnpp
Bold Commerce	BoldCommerce
BPCE	Bpce
Braintree	Braintree
BrinkCommerce	Brinkcommerce
Buckaroo	Buckaroo
Cafe24	Cafe24
CardinalCommerce	Cardinalcommerce
Cardlink	Cardlink
CCV Group	CcvGroup
Cegid	Cegid
Centra	Centra
Checkout	Checkout
Citcon	Citcon
Citibank	Citibank
ClearHaus	Clearhaus
Clearis	Clearis
CloudCart	Cloudcart
CodaPayment	Codapayment
Comerzzia	Comerzzia
Comgate	Comgate
CommerceTools	Commercetools
Commerzia	Commerzia
Computop	Computop
Conecta Software	ConectaSoftware

Platform Name	Predefined string
One.com	OneCom
OneBox	Onebox
Onil	Onil
OnlinePaymentPlatform	Onlinepaymentplatform
Openbravo	Openbravo
Opencart	Opencart
Openpay	Openpay
Optimizely	Optimizely
Oracle Commerce Cloud	OracleCommerceCloud
Orbital Revolution	OrbitalRevolution
Outpayce (part of Amadeus)	Outpayce
OxidESales	Oxidesales
P24	P24
Pacypay	Pacypay
Pay.nl	PayNI
Paybyrd	Paybyrd
Payco	Payco
PayComet	Paycomet
PayerMax	Payermax
PayGreen	Paygreen
Payletter	Payletter
Paylike	Paylike
Payline	Payline
Payment Sense	PaymentSense
Paymentwall	Paymentwall
PayNow	Paynow
Payone	Payone
Payoneer	Payoneer
PayPlug Enterprise SaaS	PayplugEnterpriseSaaS
Payrex	Payrex
PaySafe	Paysafe
Paytrim	Paytrim
PayU	Payu
Payxpert	Payxpert
PensoPay	Pensopay



Platform Name	Predefined string
Conekta	Conekta
ContentSpeed (Payten)	Contentspeed
Cosin	Cosin
Credit Agricole	CreditAgricole
Credit Mutuel	CreditMutuel
Crisp Studio	CrispStudio
CS Cart	CsCart
Cybersource	Cybersource
Danal	Danal
DanDomain	Dandomain
DAPP	Dapp
DEUNA	Deuna
Deutsche Bank	DeutscheBank
Digital River	DigitalRiver
Digital Takeout	DigitalTakeout
Dintero	Dintero
Dir&Ge	DirGe
Dlocal	Dlocal
DNA Payments	DnaPayments
e-shop 360	EShop360
Easypay	Easypay
Ebanx	Ebanx
eComm360	Ecomm360
eCommerce news	EcommerceNews
ecwid	Ecwid
Eglobal	Eglobal
EKMPOwershop	Ekmpowershop
Elavon	Elavon
Enactor	Enactor
epay	Epay
EpiServer	Episerver
Eromnet*	Eromnet
eService	Eservice
eShopWorld	Eshopworld
eStudio 34	Estudio34

Platform Name	Predefined string
PeP	Pep
PHC	Phc
PingPong	Pingpong
Pireaus Bank	PireausBank
Planet Payment (DataTrans)	PlanetPayment
Plati.online	PlatiOnline
Polcard	Polcard
PPRO	Ppro
Prestashop	Prestashop
Primavera	Primavera
Primer	Primer
ProcessOut	Processout
Prosa	Prosa
Pulse247 Oy (MyCashFlow)	Pulse247Oy
PXP Financial	PxpFinancial
Qenta	Qenta
Quickbooks	Quickbooks
Quickbutik	Quickbutik
QuickPay	Quickpay
Radial	Radial
Razer Merchant Service	RazerMerchantService
Reach	Reach
Redicom	Redicom
Redsys	Redsys
Reduncle	Reduncle
Reduniq	Reduniq
Rocket Digital	RocketDigital
Sabadell	Sabadell
Sage	Sage
Salesforce Commerce Cloud	SalesforceCommerceCloud
SAP (Hybris)	Sap
Scannet	Scannet
Scayle	Scayle
SDi Digital Group	SdiDigitalGroup
Seidor	Seidor



Platform Name	Predefined string
Eupago	Eupago
EuPlatesc	Euplatesc
Eurobank	Eurobank
eValent	Evalent
Eximbay	Eximbay
Extended	Extended
FAPS	Faps
FIK	Fik
Finqu Oy	FinquOy
Fintk Consulting	FintkConsulting
First Data	FirstData
FIS	Fis
FiServ (Clover, Telecash, AIB)	Fiserv
FlatPay	Flatpay
Floodid	Floodid
FreedomPay	Freedompay
GetNet	Getnet
GK Software	GkSoftware
Global Payments (Evo Payments)	GlobalPayments
GlobalE	Globale
GoDaddy	Godaddy
Gomag	Gomag
GoPay	Gopay
Hiberus	Hiberus
HiPay SAS	HipaySas
Hobex	Hobex
Holipay	Holipay
Hostinger	Hostinger
I'mweb	I'Mweb
ICG	Icg
Idosell/IAI	Idosellai
Ifthenpay	Ifthenpay
Ilion	Ilion
Ingenico	Ingenico

Platform Name	Predefined string
Shift4	Shift4
Shoper	Shoper
Shopify	Shopify
ShopLazza	Shoplazza
Shopline	Shopline
Shoptet	Shoptet
Shopware	Shopware
SIBS	Sibs
Simpler	Simpler
Simply.com	SimplyCom
Sipay	Sipay
Six Saferpay	SixSaferpay
Smallpay S.r.l.	SmallpaySRL
solteq	Solteq
Square	Square
Squarespace Commerce	SquarespaceCommerce
Strato	Strato
Stripe	Stripe
Studioforty9	Studioforty9
SumUp	Sumup
Swedbank Pay	SwedbankPay
TD Bank Merchant Solutions	TdBankMerchantSolutions
Telecash	Telecash
Tencent	Tencent
Thunes	Thunes
TiendaNube	Tiendanube
Tier1	Tier1
Toss Payments	TossPayments
Tpay	Tpay
Trevenque Sistemas de Información	TrevenqueSistemasDeInformacion
Trilogi	Trilogi
TrustPay	Trustpay
Twice Commerce (prev. Rentle Oy)	TwiceCommerce
Unzer	Unzer



Platform Name	Predefined string
Intershop	Intershop
Ixopay	Ixopay
J.P. Morgan	JPMorgan
Jimdo	Jimdo
JPM	Jpm
JTL Shop	JtlShop
Jumpseller	Jumpseller
KG Inicis	KgInicis
Kodmyran	Kodmyran
Kushki	Kushki
Kvanto	Kvanto
Labelgrup	Labelgrup
Legends	Legends
Lemonway	Lemonway
Lightspeed	Lightspeed
Línea Gráfica	LineaGrafica
Litium	Litium
Lloyd	Lloyd
LMS-Sport	LmsSport
Logpay	Logpay
Lyra Network	LyraNetwork
Maktagg	Maktagg
Mangopay	Mangopay
Marketpay	Marketpay
MerchantPro	Merchantpro
Microsoft Dynamics Navision	MicrosoftDynamicsNavision
MobilePay	Mobilepay
MODDO	Moddo
Mollie	Mollie
Mondido	Mondido
Monei	Monei

Platform Name	Predefined string
Upstream pay	UpstreamPay
Verifone	Verifone
Vilkas Group	VilkasGroup
Vipps	Vipps
VismaPay	Vismapay
Visualit	Visualit
VisualSoft	Visualsoft
VivaWallet	Vivawallet
VR Payment	VrPayment
Vtex	Vtex
Wallee	Wallee
Way2ecommerce	Way2Ecommerce
Weasy.io	Weasylo
Webimpacto	Webimpacto
Webmefy	Webmefy
Wells Fargo	WellsFargo
Westpay	Westpay
Wikinggruppen	Wikinggruppen
Windcave	Windcave
Wix	Wix
Wizishop	Wizishop
Woocommerce	Woocommerce
woolman	Woolman
WorldFirst	Worldfirst
Worldline	Worldline
Worldpay	Worldpay
Xsolla	Xsolla
Xsolla	Xsolla
xt commerce	XtCommerce
Young Pixel	YoungPixel
Zettle by Paypal	ZettleByPaypal

4.5.2.2 Data requirements

The Klarna-Integration-Metadata header object encapsulates information about all known integration pathways involved in a given request.

4.5.2.2.1 Integrator object

integrator provides details of the client responsible for the request, and Acquiring Partners are required to pass this object with every request.



- **Name:** The name of the integrator sending the request to Klarna. Values are not validated, but Klarna may request the value provided be adjusted to remain consistent across multiple integrators. See [Predefined platforms](#) for a list of accepted strings for defined platforms.
- **Module name:** The name of the application sending the request to Klarna. This allows Klarna to recognize different applications under the same integrator. Values are not validated.
- **Module version:** The version of the integration indicated by the `module_name` parameter.
- **Session reference:** A long-lived, unique identifier assigned by the application associated with an overall transaction or session. This identifier is stored within Klarna's logs, empowering agents to investigate issues with downstream applications.

4.5.2.2 Originators array

`originators` is an array of objects representing all applications or platforms involved in providing information included within the request. This metadata provides a detailed view of the specific software platforms, programs or modules leveraged by the Partner.

- **Name:** The name of the platform providing data passed towards Klarna in the request. Values are not validated, but Klarna may request the value provided be adjusted to remain consistent across multiple integrators. See [Predefined integrators and originators](#) for a list of accepted strings for defined partners.
- **Module name:** The name of the software module providing the data passed in the request to Klarna.
- **Module version:** An optional parameter indicating the version of the integration indicated by the `module_name` parameter.
- **Session reference:** An optional parameter containing a long-lived, unique identifier assigned by the application associated with an overall transaction or session. This identifier is stored within Klarna's logs, empowering agents to investigate issues with downstream applications.

4.5.2.4 Implementation pathways

4.5.2.4.1 WebSDK

Integrator object integration

When initializing the Klarna WebSDK, integrators can include the integrator context within the configuration object to ensure the application metadata is correctly registered:

```
Unset
const Klarna = await initiateKlarnaWebSdk({
  clientId: 'your_client_id_here',
  integrator: {
    name: 'AcquiringPartner',
    moduleName: 'SubIntegration',
    moduleVersion: '2.0',
    sessionReference: 'xxx',
  },
});
```

Originators array registration

For clients built on Klarna's WebSDK, the WebSDK exposes an interface that allows developers to register their request originator information. Plugins that use WebSDK must call this interface to register their application information. The WebSDK should include this information on every call to the server.



JavaScript

```
Klarna.appendOriginator({
  name: 'ecommerceCompany',
  sessionReference: '5555-474',
  moduleName: 'ecommercePlugin',
  moduleVersion: '1.0'
})
```

The originator metadata array will be bound to a specific instance and should be limited to no more than 6 objects, retaining the final 6 items in case of any conflict.

4.5.2.4.2 APIs

The Klarna-Integration-Metadata header object will allow for the passing of integration tagging information in all server-side requests. Although this field is technically optional, Acquiring Partners must send this information in every request towards Klarna, indicating both the version of their integration which triggers the request within the `integrator` object, in addition to any services which provide relevant data passed within the request in the `originators` array.

Java

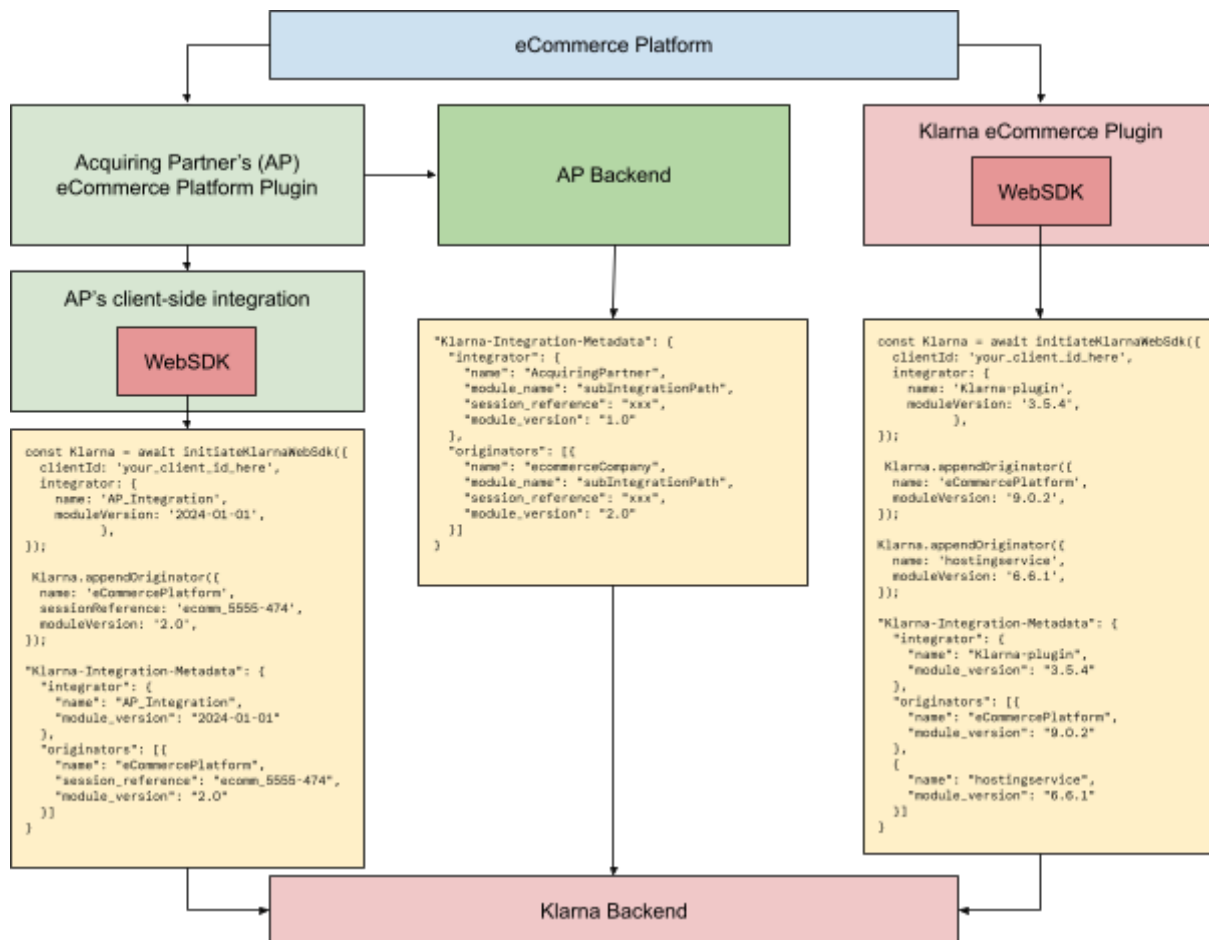
```
{
  "integrator": {
    "name": "AcquiringPartner",
    "session_reference": "xxx",
    "module_name": "subIntegrationPath",
    "module_version": "1.0"
  },
  "originators": [{
    "name": "ecommerceCompany",
    "session_reference": "xxx",
    "module_name": "subIntegrationPath",
    "module_version": "2.0"
  }]
}
```

4.5.2.4.2 Plugins and platforms

For any plugin or platform managed by an acquiring partner, partner tagging information should be included by default without additional effort from the Partner. Any plugin provisioning data directly towards Klarna should include information on the specific integration within the `integrator` object, and any plugin passing information via an acquiring partner must be included within the `originators` array - with the acquiring partner passed as the `integrator`.

As partners are required to provide this information across all integration paths, the result allows monitoring of performance across multiple integrations. Below is an example of how this complexity could be expressed in a given transaction:





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.



5. Migrating to Klarna Network

This section outlines the steps for Acquiring Partners to migrate their integration to the Klarna Network. Migration should not be attempted without previously engaging your Klarna account manager to ensure your partnership has been prepared.

5.1 Existing concepts and their future state

Review the below table to understand the relationships between legacy concepts and future state. This provides helpful context when making decisions regarding Klarna Network.

Concept	Existing Integration	Klarna Network
Endpoints	3 separate API environments (NA, EU, OC) <ul style="list-style-type: none"> Europe: https://api.klarna.com/ North America: https://api-na.klarna.com/ Oceania: https://api-oc.klarna.com/ 	Single Global API env api-global.klarna.com See: Environments
	One SDK source: <ul style="list-style-type: none"> js.klarna.com 	One SDK source: <ul style="list-style-type: none"> js-global.klarna.com See: WebSDK
Authentication	Basic auth for API No auth for JS SDK	Basic auth with empty password for API and username / token to plugin compatibility. Client id for JS SDK
	One set of credentials per subaccount per region (EU/US/AP)	One set of credentials for all Partners globally. The integrator must send in an additional account identifier to act on behalf of a subaccount.
	<MID>_<random-string>:<password> Example key: <ul style="list-style-type: none"> K123456_122343:4M08QvuEjPrtnTv6A90 	Improved descriptiveness in credentials: klarna_<test live>_<client api>_<key> Example key/id: <ul style="list-style-type: none"> klarna_live_client_UDVO...TlpWTz klarna_live_api_e1ZGI1B...5dHBIR See: API Authentication Standards

5.2 Phase 1: Migrate your Onboarding processes

What to cover:

You can onboard a new merchant and they can use the old APIs, so step 1 is integrating the account management flows

Pricing	Configured within Klarna internal systems, not accessible externally	Price Plans define the pricing rates for transactions based on the Merchant Category Code (MCC) and market specifics. Each price plan includes rates that can vary depending on the market, payment program, and channel type. Klarna creates and maintains these price plans. See: Price Plans .
---------	--	--

5.3 Phase 2: Migrate your active accounts

5.3.1 Step 1: Initiate migration

Note: If a merchant is in a suspended state it should not be migrated. If the merchant is still active, initiate the appeal process and only migrate the account once resolved.

Use the migration endpoint in Klarna Network:

- Endpoint: /v2/distribution/migrate
- Method: POST
- Payload: Submit a list containing MCC, and legacy MID. More information available [here](#).
- Limit: Endpoint supports one MID migration at a time and is idempotent if an idempotency-key is provided.

Release note:

 *Currently MIDs can only be migrated to accounts with a 1:1 relationship via this endpoint.*

5.3.1.1 Example payload:

```
JavaScript
{
  "account_reference": "M123786123412",
  "account_name": "John Doe Stakehouse",
  "account_owner": {
    "given_name": "John",
    "family_name": "Doe",
    "email": "john.doe@example.com",
    "phone": "+18445527621"
  },
  "products": [
    {
      "merchant_id": "K123456",
      "type": "PAYMENT",
      "merchant_category_code": "5411"
    }
  ]
}
```

```

    }
  ],
  "channel": {
    "websites": [
      {
        "urls": [
          "https://example.com"
        ]
      }
    ]
  }
}

```

Once the migration process is complete, the API will return a payload confirming the creation of a new account ID for each merchant. The response will include:

- Legacy Merchant ID
- New Account ID
- State (e.g., "PARTIALLY_OPERATIONAL", for more information see [Account lifecycle webhooks](#))

5.3.1.2 Example response:

```

JavaScript
{
  "account_id": "krn:partner:global:account:live:LWT2XJSE",
  "account_reference": "M123786123412",
  "account_name": "Merchant ABC US",
  "state": "PARTIALLY_OPERATIONAL",
  "state_reason": "INITIAL_SETUP"
}

```

5.3.1.3 Error handling

To ensure accounts are correctly migrated, ensure you have robust error handling if a given migration fails. Common causes for failure include:

- Attempting to migrate a suspended account.

```

response_status: 400,
error_message: Cannot migrate, merchant id %s is currently non-operational.

```

- Attempting to migrate an account that is not associated with your Klarna relationship.

```

response_status: 400
error_message: Cannot migrate, merchant id %s seems to be invalid.

```

- Attempting to migrate an already migrated account.

```

response_status: 400
error_message: Cannot migrate, merchant id %s is already onboarded in this API

```

- Attempting to migrate an account that is not on a default offering

```
response_status: 400  
error_message: Cannot migrate, merchant id %s xyz
```

In case any of these responses are received, they should be logged and handled manually as part of your migration.

5.3.2 Step 2: Confirm data and update internal systems.

Ensure that the newly created Account ID is updated in your systems to replace the legacy account credentials. After a grace period, the legacy account credentials will be disabled, and the Account ID will be used for all transactions and management within the Klarna Network.

Merchant accounts that have successfully migrated will automatically be enabled to transact just like a newly onboarded account, with no changes to pricing, settlements, or customer offerings during the migration process.

Legacy merchant credentials will be disabled if unused for 2 months. After 10 months of inactivity they will be deleted.

