Authorize.Net CIM: User Manual

Version 6.0 – For Magento® 2.4.4+ – Updated 2026-06-17

Table of Contents

Installation

The installation process differs based on where you purchased our extension.

By composer (strongly recommended)

If you purchased from Adobe Commerce Marketplace, this installation method is mandatory. No purchase is necessary though.

Step 1: Install

We strongly recommend installing, configuring, and testing all extensions on a development website before installing and using them in production.

In SSH, from your site root, run the following commands:

```
composer require paradoxlabs/authnetcim
php bin/magento setup:upgrade
```

If your site is in production mode, you will need to run these commands to recompile:

```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy -j 12
```

These commands should load and install the latest extension packages.

Step 2: Configure

See the configuration section below.

By download from store.paradoxlabs.com

NOTE: This download installation applies **only** to purchases from the ParadoxLabs Store. Everyone (including Marketplace purchases) can use the composer installation directions above.

Step 1: Upload files

Upload all files within the **upload** folder into the root directory of Magento.

Folder in Download		Folder on Server
/upload/app/	→	/app/

Step 2: Run Installation

In SSH, from your site root, run the following commands:

```
php bin/magento setup:upgrade
```

These will enable the module, flush the cache, and trigger the installation process to run.

If your site is in production mode, you will also need to run these commands to recompile sources:

```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy -j 12
```

Step 3: Configure

See the configuration section below.

Updating Authorize.Net CIM

All extension updates are free. Just follow these directions to update to the latest version.

If you installed by composer (or purchased from Adobe Commerce Marketplace)

Note, installing/upgrading this extension requires familiarity with your server's command line.

If you installed with composer, you can update using the following commands, in SSH at your site root:

```
composer update paradoxlabs/*
php bin/magento setup:upgrade
```

This will download and update to the latest extension version compatible with your system.

If your site is in production mode, you will also need to run these commands to recompile sources:

```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy -j 12
```

If you purchased from store.paradoxlabs.com

Step 1: Upload files

Log into your account at store.paradoxlabs.com and download the latest version.

Open the extension archive and extract it onto your computer.

Upload all files within the **upload** folder into the root directory of Magento.

Folder in Download		Folder on Server
/upload/app/	→	/app/

Step 2: Run Update

In SSH, from your site root, run the following commands:

```
php bin/magento setup:upgrade
```

If your site is in production mode, you will also need to run these commands to recompile sources:

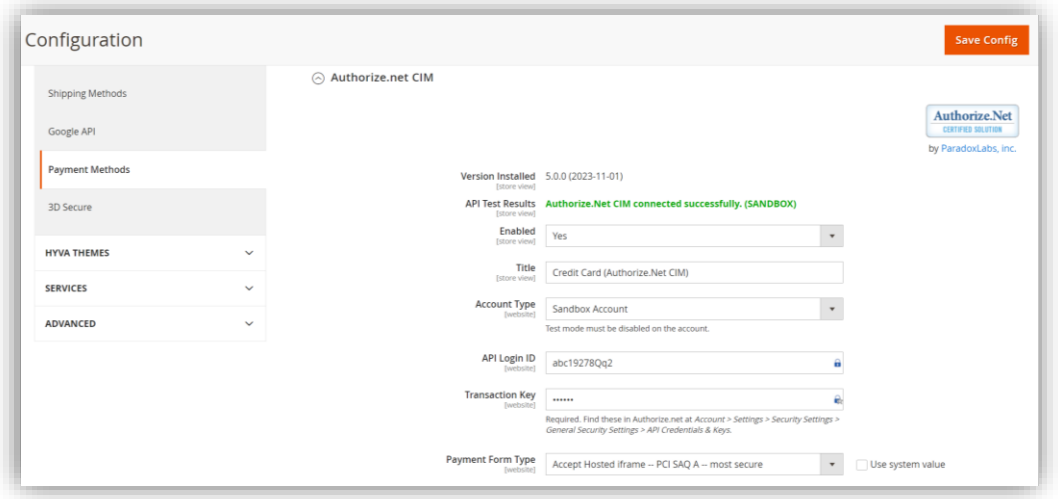
```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy -j 12
```

Configuration

Before proceeding: Sign up for an Authorize.Net merchant account if you don't have one already, and ensure your account has Customer Information Manager (CIM) enabled.

Open your Admin Panel and go to **Admin > Stores > Configuration > Sales > Payment Methods**. Toward the bottom of the page, you'll find an 'Authorize.Net CIM' settings section like the below.

General



- **Version Installed**: This is the currently installed version of our extension.
- **API Test Results**: If you've entered an API Login ID and Transaction Key, we will automatically connect to Authorize.net to verify that the API works successfully. If everything works, it will show 'Authorize.Net CIM connected successfully.' Failure can happen for several reasons including:
    - Your server can't connect to Authorize.net
    - The entered API Login ID or Transaction Key is incorrect
    - The Account Type is incorrect
    - Your Authorize.net account does not have the CIM service enabled

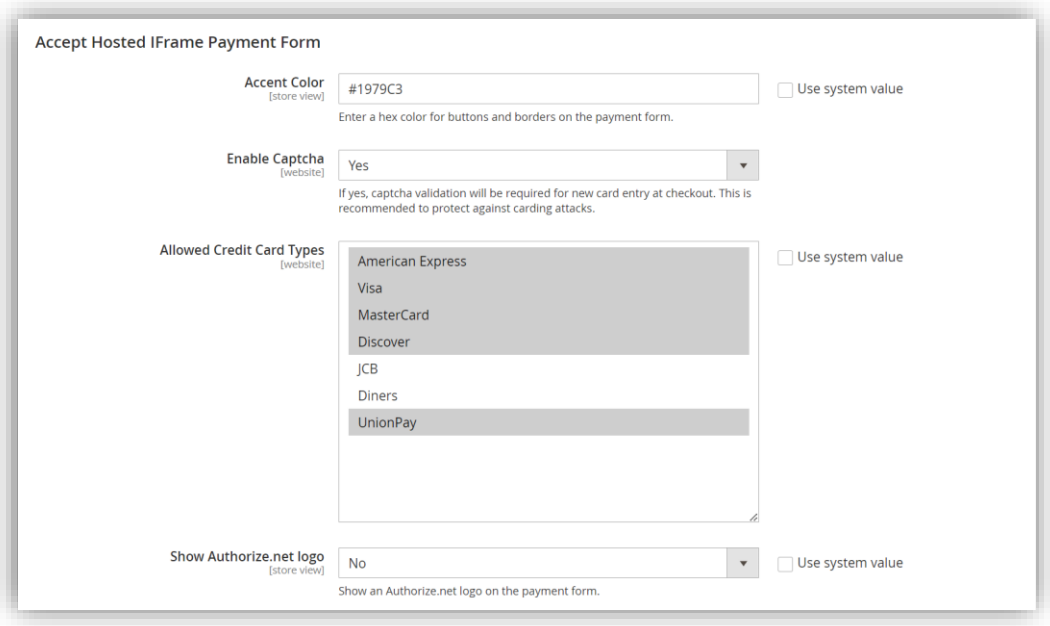
  If you get an error message, correct the error and try again.
- **Enable**: Yes to enable the payment method. If disabled, you will still be able to invoice/refund existing orders, but it will not show up as a payment option during checkout.
- **Title**: This controls the payment option label on checkout and order confirmations.
- **Account Type**: Choose 'Live Merchant Account' if the API Login ID and Transaction Key you entered are for a merchant account, even if you have test mode enabled. If this value is not correct, the API Test Results will report *'Your API credentials are invalid.'*
    - If you want to test payments, you must have a separate sandbox account. You can create one free here: https://developer.authorize.net/hello_world/sandbox/
- **API Login ID**: This is a value from your Authorize.net account. To find it, log into your Authorize.net account, then go to **Account > Settings > Security Settings > General Security Settings > API Credentials & Keys**. Your API Login ID will be in the middle of the page.
- **Transaction Key**: This is a secret value from your Authorize.net account. If you don't know it, log into your Authorize.net account, then go to the same page as API Login ID above. You will have to enter your secret

answer to generate a new transaction key. Record the generated value for your records.
WARNING: Generating a new transaction key will cause your existing transaction key to expire 24 hours later. If any other services or software are connected to your Authorize.net account, you MUST update them with the new transaction key immediately.

- **Payment Form Type**: Choose one of the following
    - **Accept Hosted iframe** – PCI SAQ A – most secure (recommended)
    - **Accept.js tokenization** – PCI SAQ A-EP
    - **Inline form** – PCI SAQ D – will be removed in the future

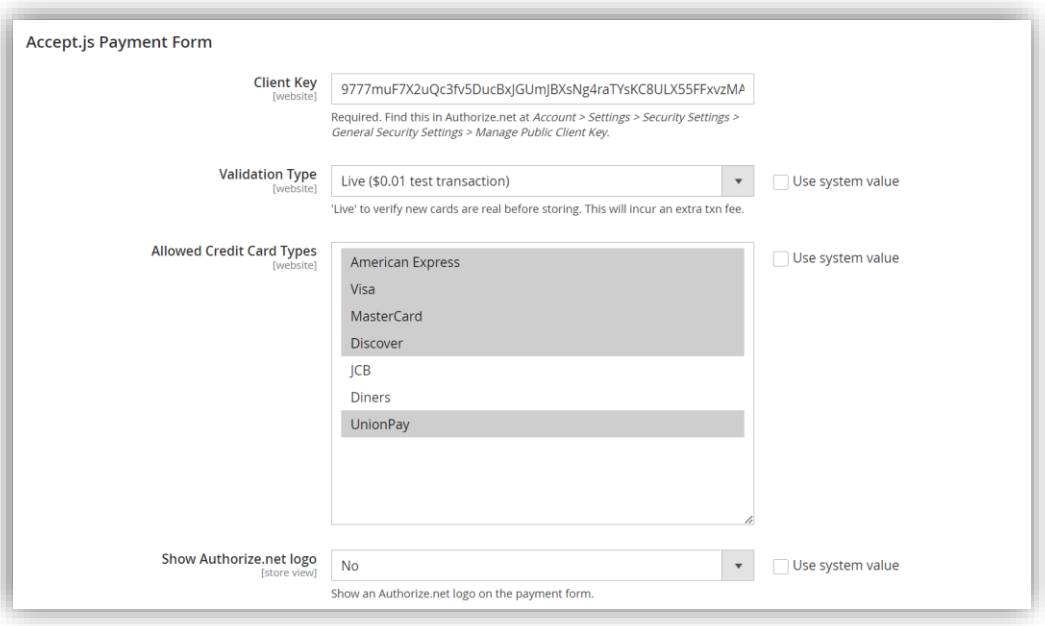
We strongly recommend using the Accept Hosted payment form, unless you need direct control over the form style or behavior, or have integrations that depend on Accept.js. Your choice will affect which settings appear next:

Accept Hosted IFrame Payment Form



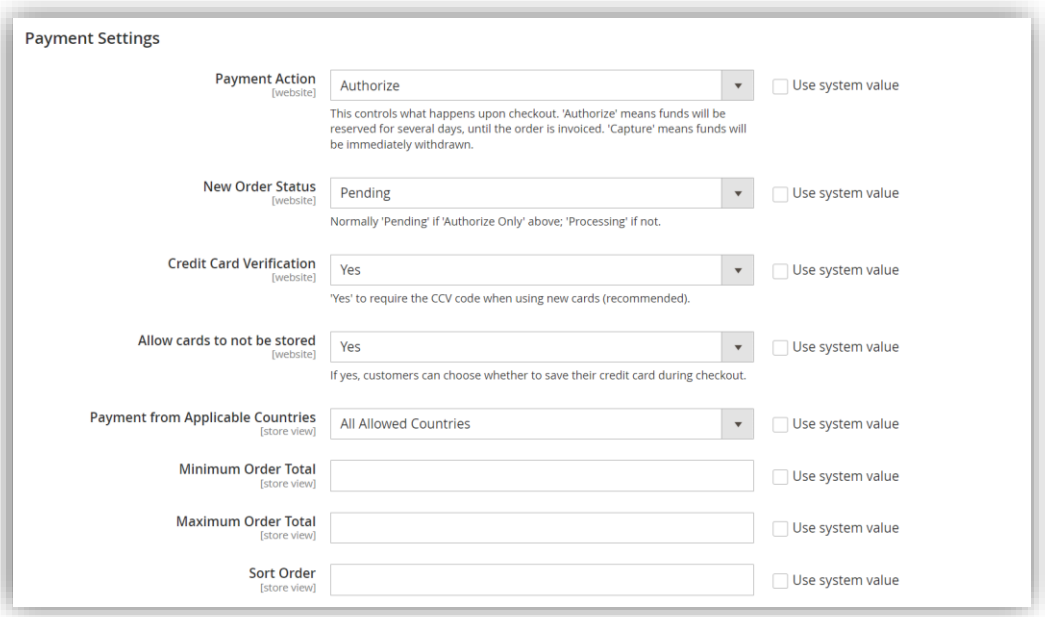
- **Accent Color**: This will change the color of buttons and highlights on the credit card form. Enter a hex color that matches your checkout page or brand colors. The default is #1979C3, a medium blue.
- **Enable Captcha**: Yes enables an inline reCAPTCHA on the hosted credit card form. We strongly recommend enabling this to combat card testing.
- **Allow Credit Card Types**: Choose the CC types you want to allow on checkout. Please set these to match your allowed credit card types in Authorize.net, to avoid surprising users.
- **Show Authorize.Net Logo**: If yes, checkout will display an 'Authorize.Net' logo next to the payment form.

Accept.js Payment Form



- **Client Key**: In order to use Accept.js, you must enter your Client Key. To find this, log in to Authorize.Net and go to **Account > Settings > Security Settings > General Security Settings > Manage Public Client Key**.
- **Allow Credit Card Types**: Choose the CC types you want to allow on checkout.
- **Show Authorize.Net Logo**: If yes, checkout will display an 'Authorize.Net' logo next to the payment form.

Payment Settings

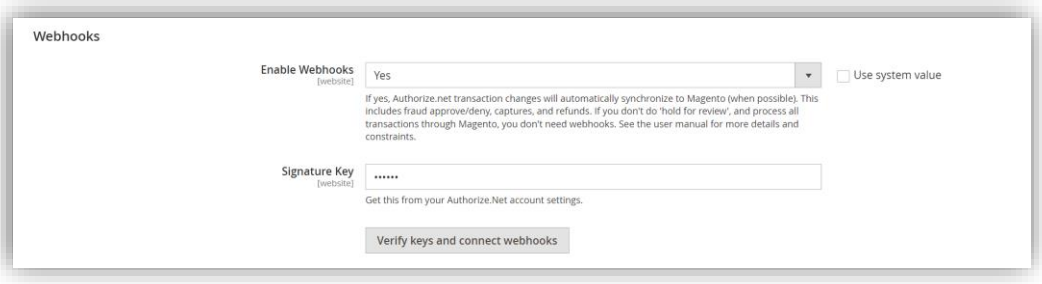


- **Payment Action**: Choose from the following options.

  - **Save info (do not authorize)**: This will require customers to enter a credit card on checkout, and store that credit card in Authorize.net CIM. No funds will be captured or held from the credit card upon checkout. When you invoice the order, that will run a standalone authorize+capture transaction, but it is not guaranteed to go through. This can be useful for merchants that have a long delay between order and invoicing, or where the final billing amount can vary a lot.
  - **Authorize**: This will authorize the order amount upon checkout, allowing for later invoicing and capture of the funds. The order amount will be reserved (held) for several days. If you do not invoice within a couple weeks, the authorization will expire, and invoicing will perform a standalone authorize+capture transaction instead (which is not guaranteed to go through). This is recommended when it will take more than three days to fulfill your orders.
  - **Authorize and Capture**: This will capture all funds immediately when an order is placed. This is the easiest for payment processing, as long as you consistently complete orders within three days of checkout.
- **New Order Status**: Set this to your desired initial status for orders paid by Authorize.net CIM. Default Magento behavior is 'Pending' for Authorize payment action, and 'Processing' for Authorize and Capture.
- **Validation Type**: Choose from the following options.
  - **Validate fields and expiration date**: In this mode, Authorize.net will verify that the credit card number and expiration date are possible, but will not talk to the card processor. The card details and/or billing address may be invalid.
  - **Run $0.01 test transaction first**: This will run a test transaction against the credit card to verify that all details (card number, expiration date, CCV, AVS) are correct. The transaction amount depends on the card type. The transaction is immediately voided if successful; the customer will never see it on any statements. However, this does incur an additional transaction fee on your account.
    The benefit of live validation is that all cards stored in Authorize.Net CIM (and visible on customers' accounts) are guaranteed to be valid and usable.
    The downsides are the extra validation fee, and a chance of 'duplicate transaction' errors if customers enter part of their card info incorrectly.
- **Credit Card Verification**: If yes, customers will be prompted for their credit card security code when entering a new card. Strongly recommended.
- **Allow card to not be stored**: If yes, customers will have a 'Save for next time' checkbox on checkout. If no, logged in customers will see a message instead: *"For your convenience, this data will be stored securely by our payment processor."* Guests will never see the option to store a credit card.
  Note that all cards are always stored in CIM, regardless of this setting or the customer's choice. This is necessary for payment processing. If the order is placed as a guest, or the customer chooses to not save their card, it will be automatically purged from all systems after the maximum refund period is past. This ensures the info is available for edits, captures, and refunds, but respects the customer's wishes.
  If a card is 'not saved', it will not display under the customer's saved credit cards (Account > My Payment Data), nor will it be selectable during checkout. Note that as an admin, order 'edit' or 'reorder' will bypass this, always allowing reuse of the original payment info (unless it was since purged).
- **Payment from Applicable Countries**: This setting allows you to limit which countries are able to use the payment method.
- **Minimum Order Total**: This setting allows you to set a minimum order value for the payment method. For instance, set to 5 to only allow credit card checkout for orders of $5 or more.
- **Maximum Order Total**: This setting allows you to set a maximum order value for the payment method. For instance, set to 1000 to only allow credit card checkout for orders of $1000 or less.

- **Set Order**: This setting allows you to change the order of payment options on checkout. Enter a number for this and all other payment methods according to the order you want them to display in.

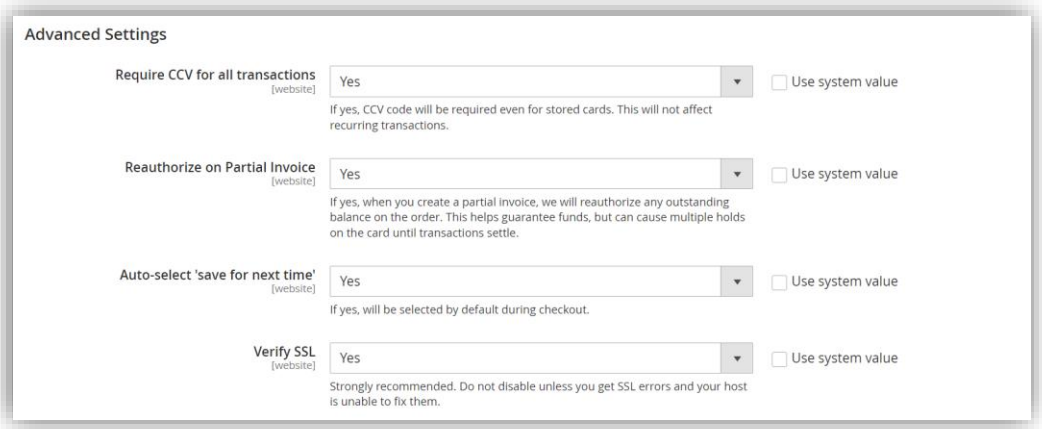
Webhooks



- **Enable Webhooks**: If yes, Authorize.net transaction changes will automatically synchronize to Magento (when possible). This includes fraud approve/deny, captures, and refunds. This is only necessary if you process approvals, denials, captures, or refunds through an external system (like an ERP or order manager, or Authorize.net directly).
  Note that this cannot synchronize partial invoices or partial refunds. Invoices and refunds will only be synced to Magento by webhook if they are for the full order amount; others should be reconciled manually or by API.
- **Signature Key**: In order to use webhooks, you must enter your Signature Key. To find this, log in to Authorize.Net and go to **Account > Settings > Security Settings > General Security Settings > API Credentials & Keys**. Under 'Create New Key(s)', select New Signature Key, then proceed through the form until you're given the key. Record the generated value for your records.
  WARNING: Generating a new signature key will cause your existing signature key to expire 24 hours later. If any other services or software are connected to your Authorize.Net account and use this key, you MUST update them with the new signature key immediately.

Once you've entered your signature key, click **Verify keys and connect webhooks** to complete the webhook setup, then save the settings.

Advanced Settings



- **Require CCV for all transactions**: If yes, customers and admins will be required to reenter the credit card security code when paying with a stored card.
- **Reauthorize on Partial Invoice**: If yes, a new authorization for the remaining balance will be created when you invoice part of an order. This helps guarantee funds, but can cause multiple holds on the card until transactions settle. Any failure during reauthorization is ignored.
- **Use multiple CIM customer profiles**: If yes, a new Authorize.Net CIM customer profile is created each time a customer saves a new card, rather than adding every card to a single profile. This prevents "You cannot add more than 10 payment profiles" errors for customers who order frequently. Default is no. Hosted forms always use multiple profiles, regardless of this setting.
- **Auto-select 'save for next time'**: If yes, the 'save this card for next time' checkbox will be checked by default (opt-out). If no, customers will have to explicitly opt in to store and reuse their card.
- **Verify SSL**: If yes, the Authorize.Net API connection will be verified against known SSL information for the API. Do not disable this unless you encounter SSL errors from the API test results and your host is unable to fix the underlying problem. Disabling this will make your payment processing vulnerable to MITM (man-in-the-middle) attacks.
- **Hosted Communicator URL**: If you are using a PWA or headless frontend with Accept Hosted, you must create a page on your frontend domain to pass messages from the form to your checkout. If this applies to you, see file `vendor/paradoxlabs/authnetcim/view/base/templates/hosted/communicator.phtml` for the code that page needs to contain. Otherwise, leave this blank to use the default Magento communicator URL.

ACH Processing (eCheck)

If you want to accept ACH (eCheck) payments, the first step is to apply and be approved by Authorize.Net and eCheck.Net for ACH processing. Your account must be approved, or ACH payments will not go through.

If/when your account is approved for ACH processing, then complete the payment method settings for '**Authorize.Net CIM – ACH (eCheck)**'. This is a separate payment method option, with its own settings.

All settings are analogous to the credit card settings covered above.

Note that there are significant differences in how ACH payments work compared to credit cards. Although all processes appear the same to Magento, when and how money is moved is quite different. We strongly recommend familiarizing yourself with the eCheck.Net FAQ:
https://support.authorize.net/knowledgebase/Knowledgearticle/?code=000001388

Usage

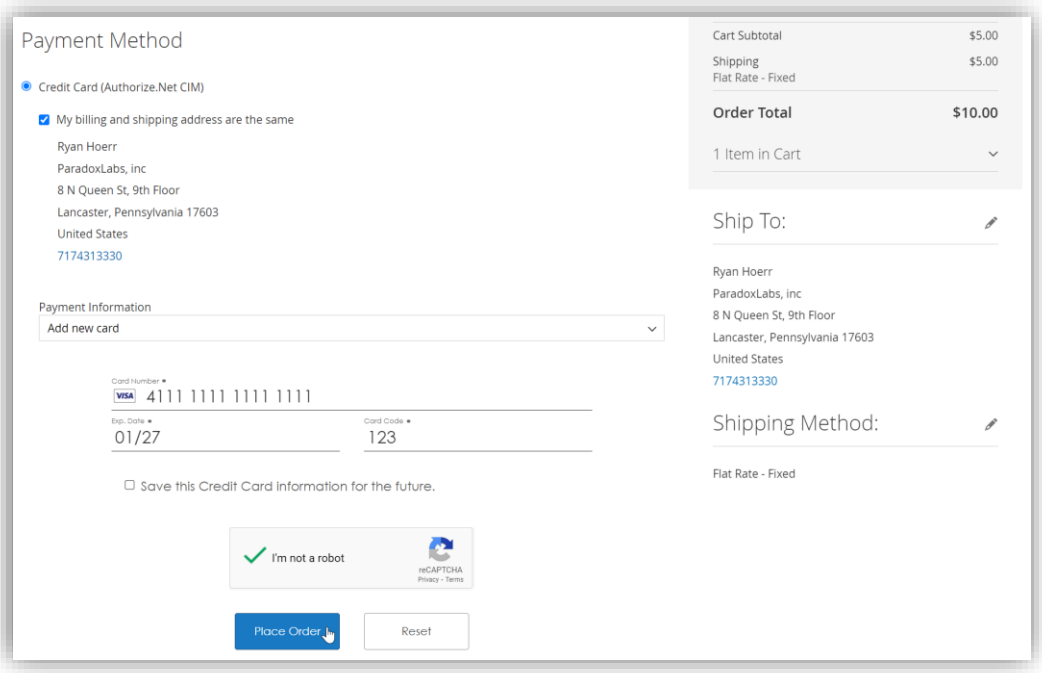
There isn't much to using this payment method, your users should find it seamless and intuitive.

Here's what you get:

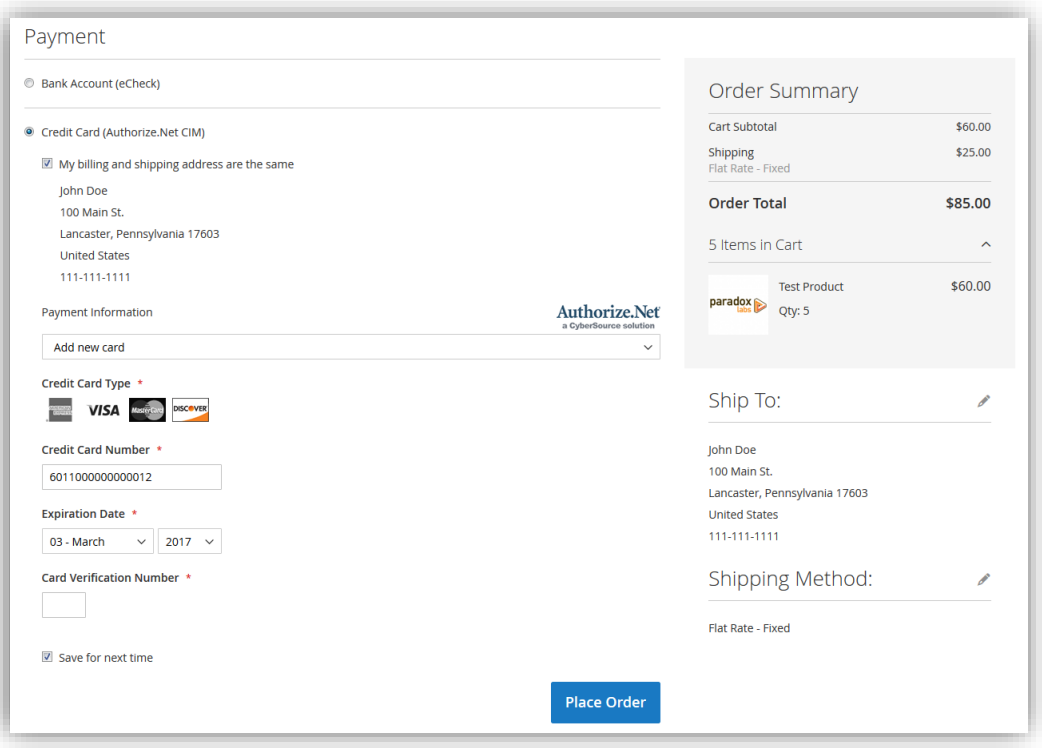
Checkout Payment Form

The frontend payment form lets you choose/enter billing address and credit card. You can choose an existing card (if any) from the dropdown, or to add a new one. Credit card type is detected automatically, and details are validated as they are entered (as much as possible).

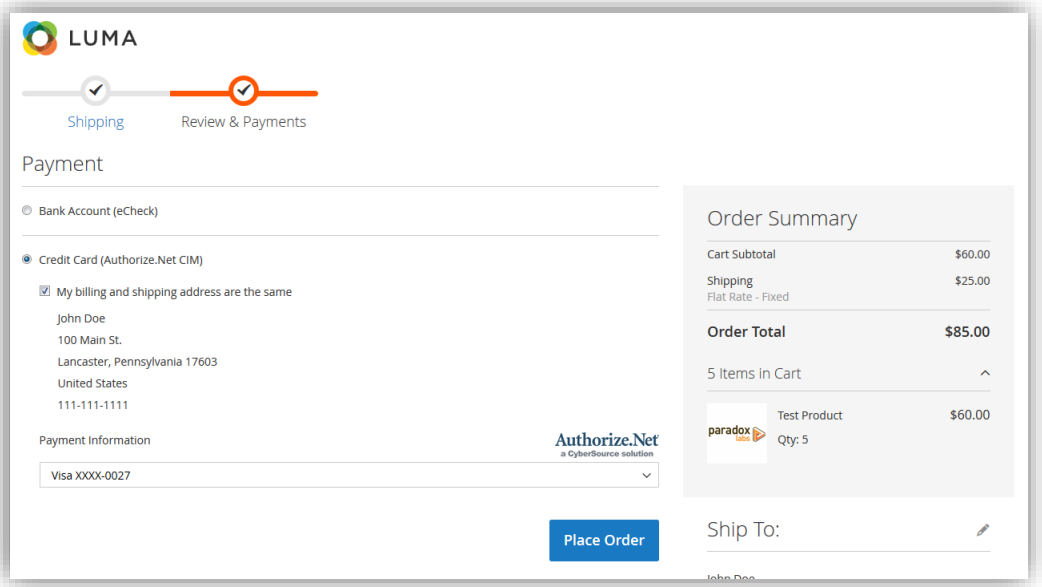
For the Accept Hosted form and default Luma checkout, the checkout page looks like:



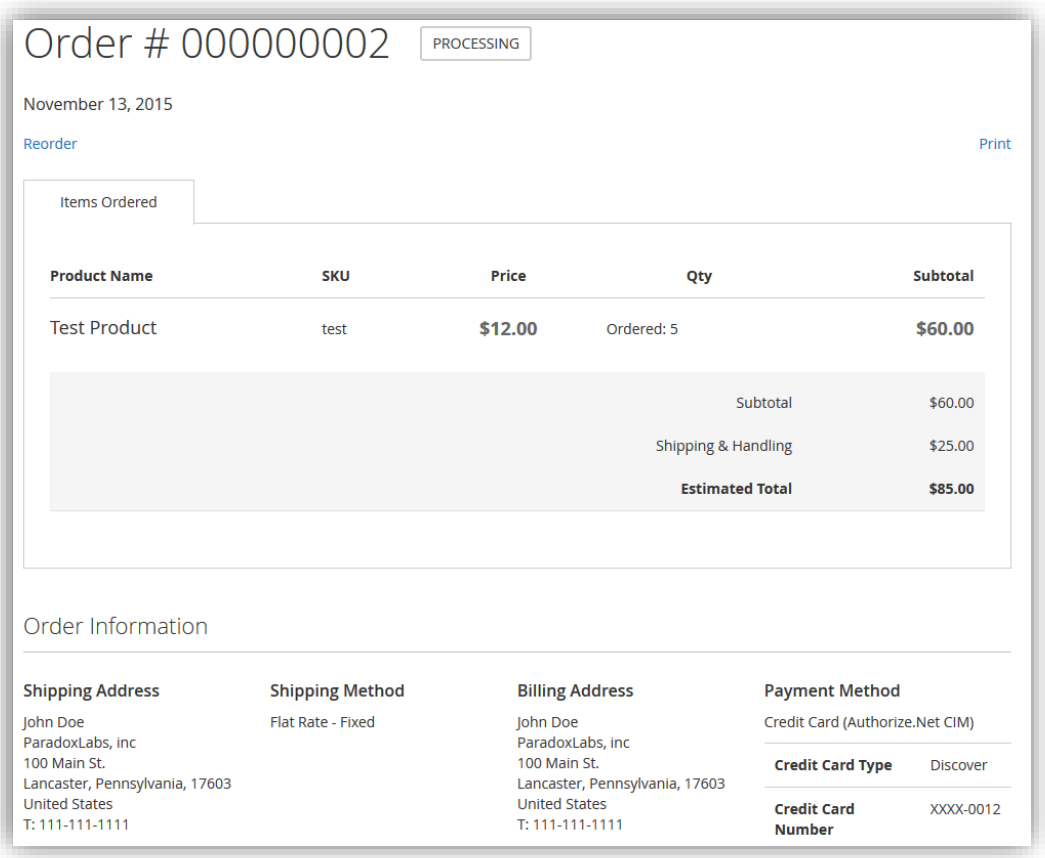
For the Accept.js form or inline form, it looks like:



If the customer has stored cards, their most recent one will be selected by default:

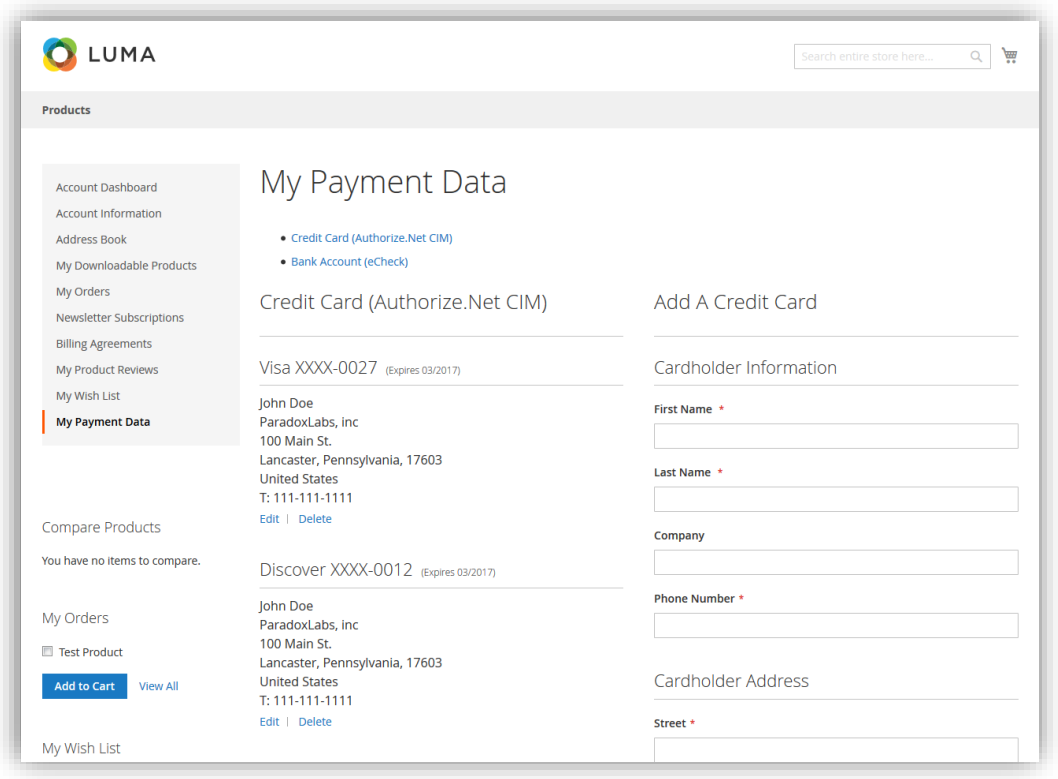


Order status page



Customer 'My Payment Data' account area

The My Payment Data section allows customers to see their stored cards, add, edit, and delete. When a customer deletes a stored card, a prompt appears to confirm the removal.



Note that cards associated with an open (uncaptured) order cannot be edited or deleted. They will display a *'Card In Use'* message in place of the buttons. As soon as all orders paid by the card are completed, the 'Edit' and 'Delete' buttons will appear.

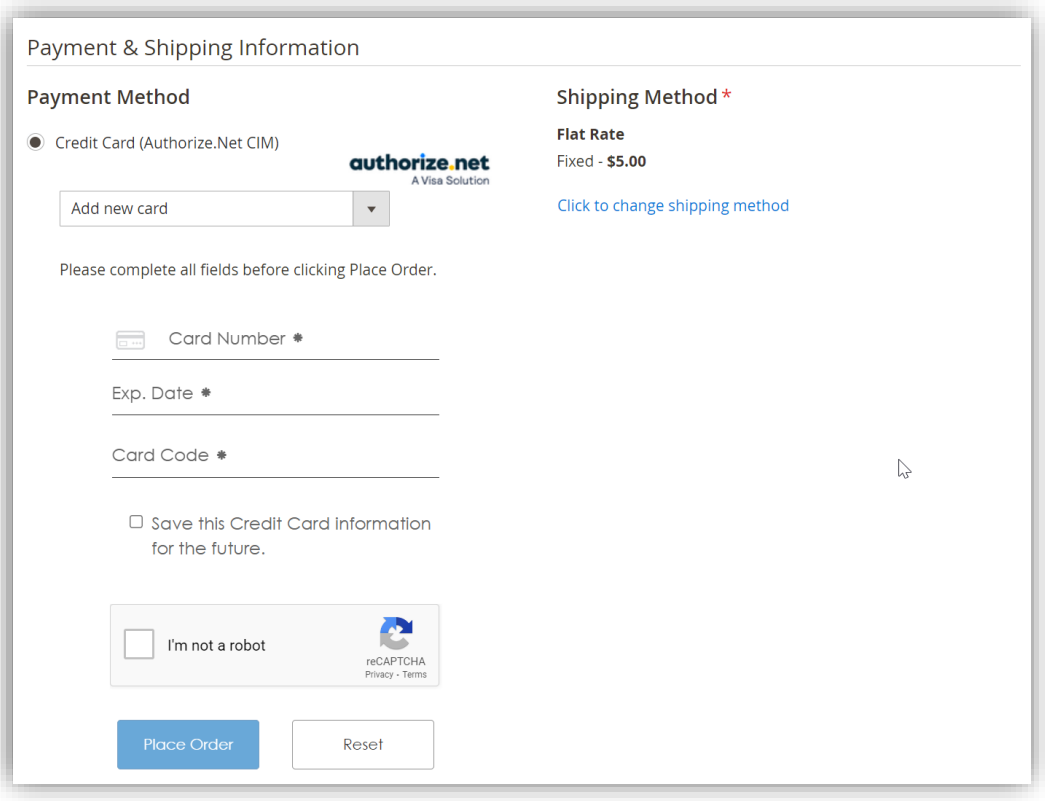
To prevent abuse, this section will only be available to customers after they have placed a successful order. If a customer attempts to access the page before then, they'll be redirected to the Account Dashboard with the message, *"My Payment Data will be available after you've placed an order."*

Also to prevent abuse, if a customer receives errors trying to save a card five times, they will be blocked from access for 24 hours with the message, *"My Payment Data is currently unavailable. Please try again later."*

These behaviors can be adjusted or disabled by configuration if necessary. To do so, go to: *Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Security Settings*

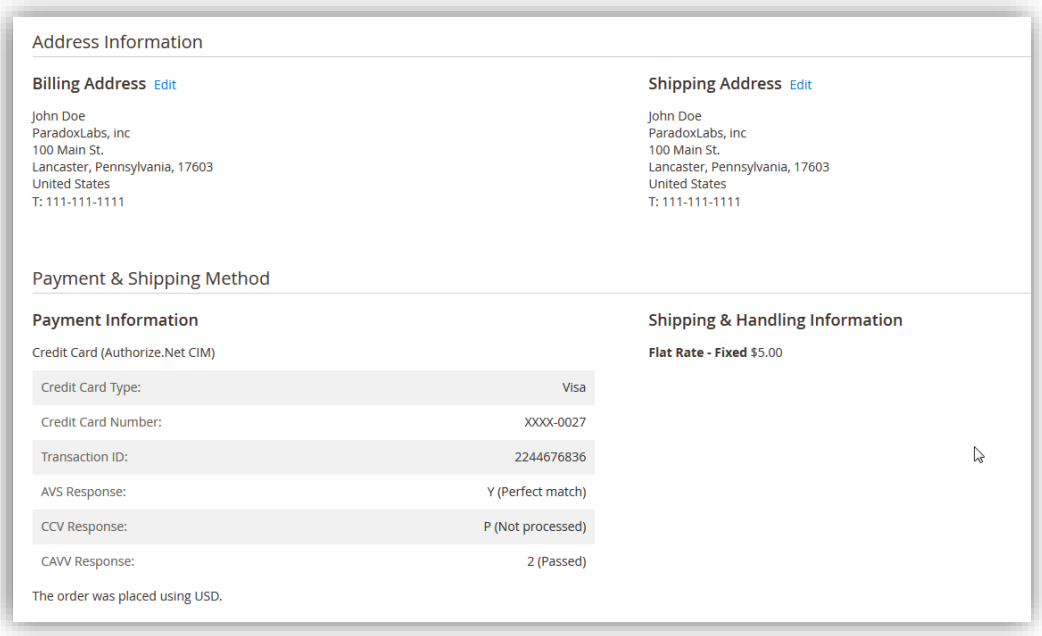
Admin order form

The admin form displays the same options as frontend checkout.



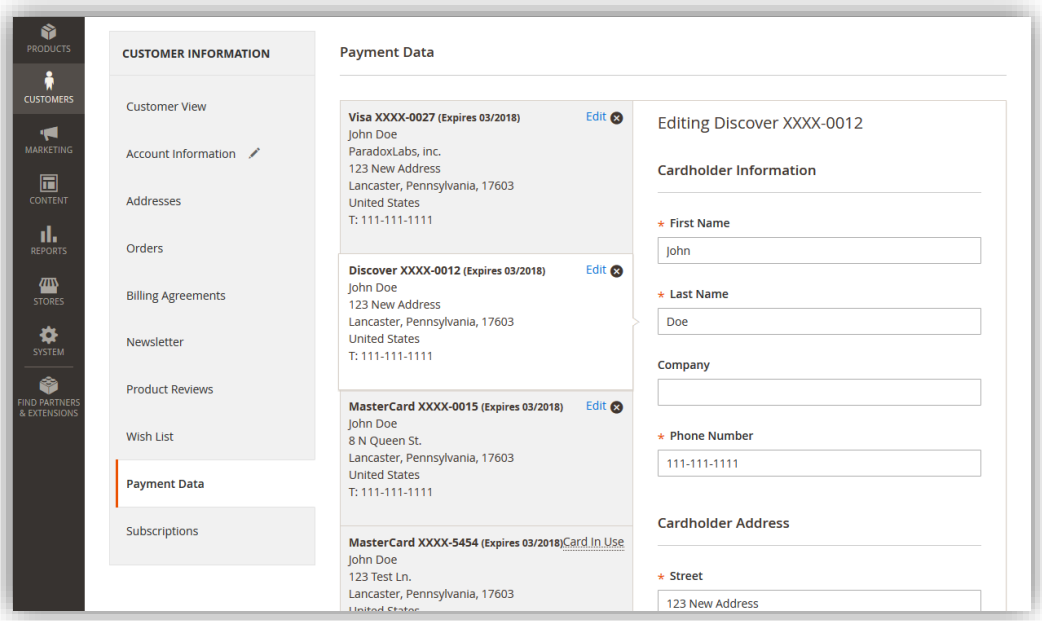
Admin order status page

The admin panel shows extended payment info after placing an order, including transaction ID and validation results. These details are not visible to the customer at any time.



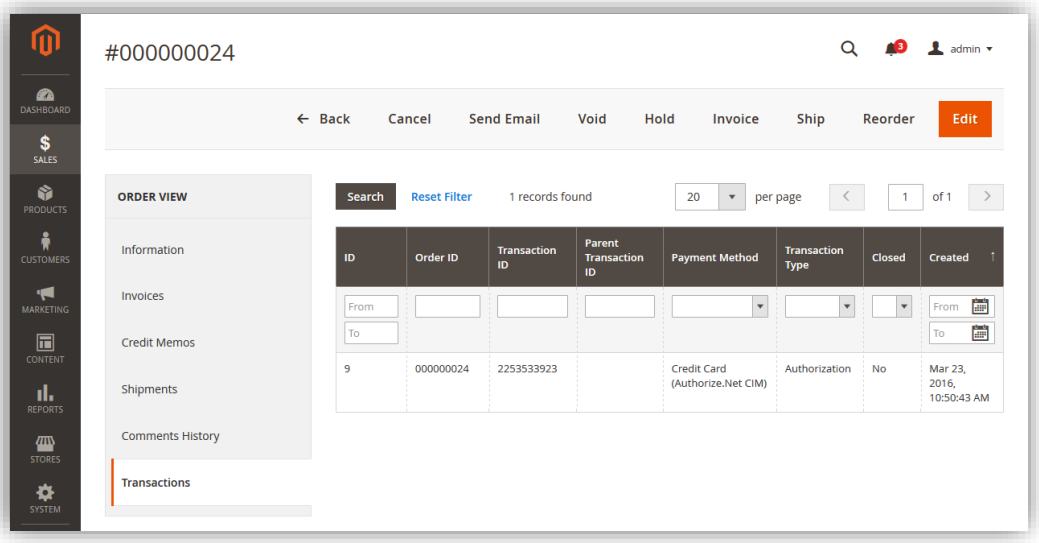
Admin customer 'Payment Data' account area

Viewing a customer, you will see an added 'Payment Data' tab. This shows all of the same information with all of the same functionality as the equivalent frontend section.

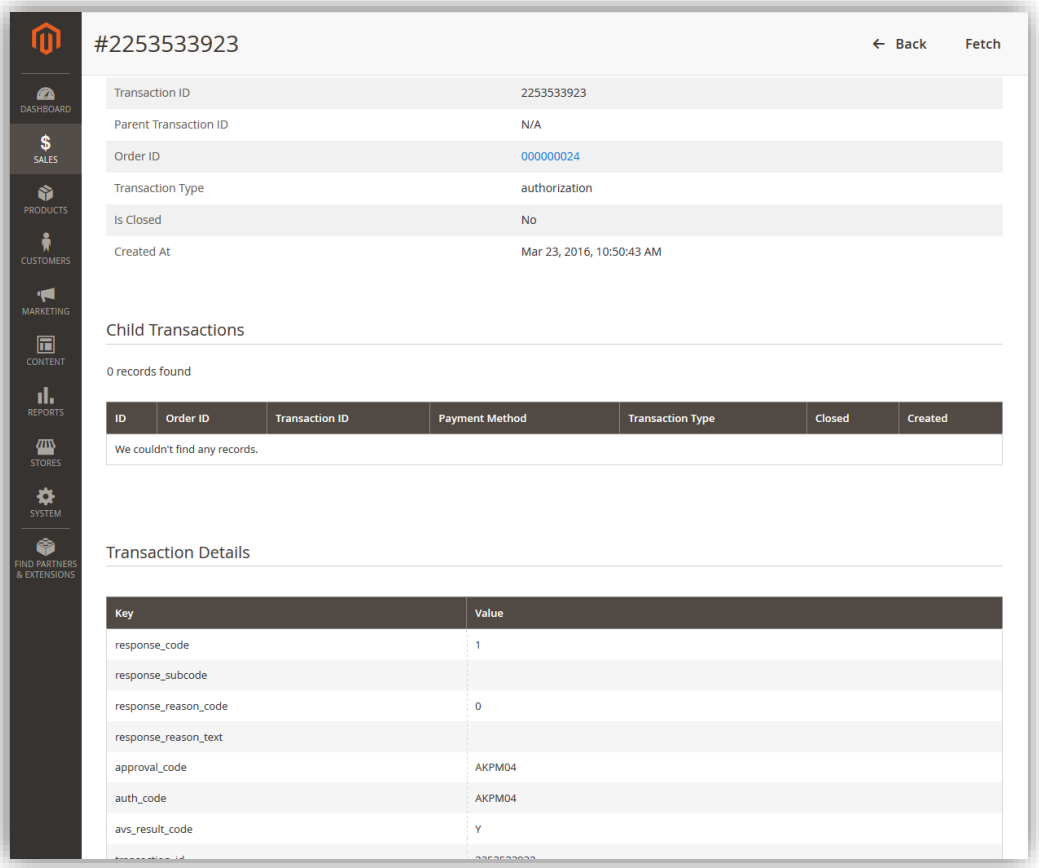


Admin transaction info

Viewing an order, you can also see full transaction info from the 'Transactions' tab.



Click into a transaction, and you'll see all of the raw transaction data from Authorize.Net.



Account Updater

This extension includes support for Authorize.net Account Updater. If your Authorize.net account has Account Updater enabled, the extension will automatically apply card changes on the first of each month. This includes new card number, new expiration date, and closed account.

Please ensure your Magento installation has cron running smoothly in order for this to work as expected.

For more information on Account Updater, including how to enable it, see:
https://support.authorize.net/knowledgebase/Knowledgearticle/?code=000001022

Be aware there are transaction fees associated with using the Account Updater service.

Form Types, Security, and Frequent Questions

Accept Hosted iframe

How does it work?

The Accept Hosted payment form type uses an iframe to load the entire payment form (credit card number, expiration date, and CVV) on a page straight from Authorize.net. Because it's an Authorize.net controlled page, your site (and any possible malicious scripts) have no ability to access that payment form or anything the customer enters into it. The info goes straight to Authorize.net, and then they report back the results to your site for order processing.

From a technical standpoint: When you load the payment form, your website frontend requests a form token from your server, and your server initiates a secure payment session with Authorize.net. This ensures the form parameters (merchant, customer, payment amount) cannot be altered in any way. That produces a short lived token, which we then use to load the hosted form. Every time an event happens with form, it passes a message to your checkout page through a 'communicator'. The checkout page JS then handles each event with an appropriate response (like placing the completed order, or showing an error message, or resizing the payment form).

This contrasts with the Accept.js form type, where the payment form fields are directly within your checkout page, but transmit the data to Authorize.net through a JavaScript library; and contrasts with the 'inline form' type, where the payment form fields are directly within your checkout page and transmit the credit card data straight to your server and then to the Authorize.net API.

Why use it?

The Accept Hosted form is the most secure Authorize.net payment form possible. Because all payment collection is offloaded to Authorize.net, this form type is eligible for PCI SAQ A compliance.

Common issues and behavior notes

The important thing to note is the hosted form means your website has less control:

- You can't customize the payment form beyond basic styling;
- Magento won't be notified of or log transaction failures;
- Any code or API errors that occur on your checkout page or during order processing may prevent successful ordering.

With the hosted payment form, and 'authorize' or 'capture' payment action, clicking the button within the form immediately runs a payment authorization with Authorize.net. If that transaction succeeds, the success status is passed to your checkout page (via the JS event messaging 'communicator'), and then the checkout page submits the order to Magento (like clicking 'Place Order'). This means the payment is processed (creating an authorization) before the order actually exists in Magento.

If your payment action is 'capture', the funds are captured after the order exists in Magento, making it a two-step process. This prevents any risk of money being taken without having a corresponding order.

This process introduces two new failure modes compared to the other form types:

1. The Authorize.net form could fail to communicate with your checkout page. If this happens, we have no way to know when the customer completes and submits the payment form. This can happen because of a JS error, or because of a "X-Frame-Options" header on the communicator blocking it from being loaded.

2. The checkout form or server (Magento) could fail to complete and save the order successfully. If this happens, no order will be created for their authorized payment transaction. The customer may or may not receive an error message on checkout. This can happen because of some JS blocking checkout 'place order', or because of some code in Magento failing the order submission (usually API integrations). In normal circumstances this would result in an error message and no order. With the hosted form, it results in an error message, no order, and an orphaned authorization transaction.

More details on each case and possible solutions:

Authorize.Net CIM: Payment gateway failed to connect, or "No message received from communicator"
If you are using Hosted Forms in our Authorize.Net CIM module you may encounter this error. This message means we haven't gotten any communication from Authorize.Net within 30 seconds from the iframe being initialized.

If you need assistance with this error, please open a ticket, and include access to your checkout and steps to reproduce.

Possible Causes

- Your site is forcing a 'X-Frame-Options' header on the communicator.
- The payment iframe failed to load.
- A JavaScript error on the page prevented script processing.
- Your website does not have SSL on all pages.
- The communicator has a different origin domain than your checkout page.

Determining the exact cause will require reviewing the browser console and network requests when it happens.

X-Frame-Options

The most likely explanation by far is the 'X-Frame-Options' header being set to 'SAMEORIGIN' or 'DENY' for the communicator page request.

If this is the cause, opening the browser dev tools console and reloading the payment form will show a console error like:

```
Refused to display 'https://yoursite.com/.../communicator/' in a frame because it set 'X-Frame-Options' to
'sameorigin'.
```

You have two options if so:

1. Talk to your dev team or hosting provider to **find and remove any service configurations that are overriding the X-Frame-Options header for your site**. Magento also includes this header for Apache by default in pub/.htaccess. The proper configuration is to disable any overrides, and enable the header in Magento configuration so that Magento is outputting it directly. At that point, our extension code can disable it for the form communicator page only, and the Hosted payment form can work as intended.

   Any service that touches your web requests could be involved. Be sure to check:
   a. Your server software: Apache, Nginx, Varnish, etc.

b. Security and transparent CDN services: Cloudflare, Fastly, Sucuri, etc.

2. If you are unable to find or disable the X-Frame-Options header, you will have to switch to the Accept.js payment form instead of Accept Hosted. Please do that in Magento configuration: **Admin > Stores > Configuration > Sales > Payment Methods > Authorize.net CIM > Payment Form Type: "Accept.js"**

Authorize.Net CIM: Extra authorizations when using Hosted Forms

If you switch to the new Accept Hosted iframe payment form for Authorize.Net you may notice additional transactions appearing in your dashboard. In some cases, you may notice there are no orders in the Magento Admin at all.

Additional Details

This happens because the hosted form 'Place Order' button creates an authorization before Magento performs any processing to create the order.

That means:

- There won't be a record of those authorizations in Magento's log files, because the authorization happens directly between the customer and Authorize.net;
- If any error prevents Magento from creating the order, you will end up with an authorization but no order. We try to automatically void the transaction, but depending on the type of error this is not always possible.

While the Hosted Form may make failed submissions more obvious, the authorizations will expire within a week. No funds are captured unless and until Magento creates an invoice for an order.

Solutions

Check your Magento logs for any errors or exceptions occurring at checkout during order placement. As much as possible, you want to ensure an order always goes through once the customer clicks 'Place Order', unless there's a very good reason for the order to fail.

If the excess authorizations are problematic, consider changing the payment form. This only affects the Hosted payment form. To change the form, go into the Magento admin payment settings, and set the "Payment Form Type" to Accept.js.

Accept.js tokenization

What is Accept.js, and why should I care?

Accept.js allows credit card information to be sent straight from your customers' browsers to Authorize.net, without touching your web server. Customers enter the card on your checkout form, but Authorize.net uses JS to tokenize the card and change it into a "one-time use token" before it gets sent to your web server. Your web server never sees the raw credit card number, so this improves your website's security, and reduces your PCI compliance exposure.

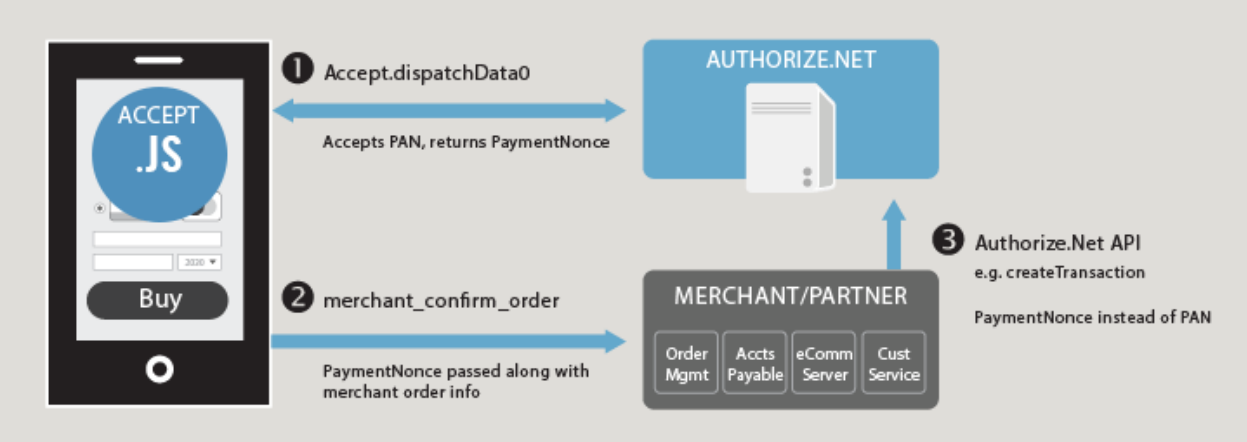


Image © 2016 Authorize.Net (used by permission)

Since Accept.js sends the credit card number directly to Authorize.Net, using our extension and Accept.js for all credit card transactions may make you eligible for PCI Self-Assessment Questionnaire (SAQ) A-EP, rather than the longer and more intensive PCI SAQ D form. For details on the SAQ types and why this is the case, see "Self-Assessment Questionnaire Instructions and Guidelines (3.2)" (PDF, by PCI Security Standard Council).

Enabling Accept.js has minimal impact on user experience. The credit card form is still located on your website, using your theme's templates and styles. The data is tokenized behind the scenes when the form is submitted. Any processing errors will be displayed to the customer in an alert window.

How do I enable Accept.js?

In order to use Accept.js, change the 'Payment Form Type' setting to 'Accept.js' at **Admin > Stores > Configuration > Sales > Payment Methods > Authorize.Net CIM**.

After changing that setting, you will see a new field for Client Key. You must enter your Client Key from Authorize.Net. To find this, log in to Authorize.Net and go to **Account > Settings > Security Settings > General Security Settings > Manage Public Client Key**.

Note that Accept.js *requires* every page with a payment form to use SSL (including your admin panel and dev sites). We strongly recommend testing checkout after enabling Accept.js. If you experience problems, please contact us.

In the interest of security, we also strongly recommend evaluating your Magento admin accounts, and only providing access to Magento's configuration area to those that absolutely require it.

Common Accept.js Errors

Accept.js is another layer of complexity on top of the existing credit card processing, and that means more things that can go wrong. Here are errors you might start seeing after you enable Accept.js, and what they mean:

"Invalid token. Please re-enter your payment info. (E00114)" (Log: 'Invalid OTS Token.')

On extension versions 4.1 and below, this error usually means Authorize.Net rejected the one-time-use payment token we sent them for the transaction. Usually this happens when the customer tries to place an order with a new credit card, their payment is rejected (AVS failed/wrong billing address, or transaction declined, etc.). After they get that error message, they immediately try hitting 'Place Order' again, without changing any of their billing or payment info. Since none of the info changed, we did not request a new token, the existing one expired, and hence the error.

How to fix: Have the customer re-enter their billing and payment info, taking care to correct whatever error they encountered originally.

OR:

Invalid token errors can also occur after changing Authorize.Net accounts (causing CIM customer profiles to disappear). For customers who've used the payment method prior to the account change, their Magento profile will have their old profile ID cached. The missing profile ID causes the initial transaction request to fail. We create a new profile automatically, but the Accept.js token was used up on the initial request, resulting in this error condition.

How to fix: Remove all CIM data associated with the old Authorize.Net account from Magento. Cached profile IDs are the major problem, but any old stored cards will also be invalid and should be removed from database table `paradoxlabs_stored_card`. You can remove the cached profile IDs with the following SQL query:

```
DELETE FROM customer_entity_varchar WHERE attribute_id=(SELECT attribute_id FROM `eav_attribute` WHERE
`attribute_code` = 'authnetcim_profile_id');
```

Please run this with the utmost caution. The query is safe, it will not have any negative impact on your site or the payment method other than causing some extra API requests temporarily, but any manual SQL operations should be handled very carefully.

"We did not receive the expected Accept.js data. Please verify payment details and try again. If you get this error twice, contact support."

This error means that your server received a raw credit card number, instead of the Accept.js token that was expected. We will never accept raw credit card details with Accept.js enabled. This means either the checkout or payment form that was submitted is incompatible with Accept.js, or the customer somehow completed the form without triggering Accept.js.

How to fix: If most customers are able to checkout successfully, collect info about the customer's browser and operating system, then either have them re-attempt checkout (possibly using another browser or device), or assist them by placing an admin order instead.

If no customers are able to complete checkout, or you can reproduce the problem yourself, please disable Accept.js temporarily, then contact us for support.

Browser console displays "Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://js.authorize.net/v1/AcceptCore.js." (or similar message) on checkout

This message happens because your webpage is unable to access that file directly. Usually this would be a problem (hence the warning message), but in this case this is actually normal and intended behavior. This is part of the many security features of Accept.js: Your site cannot (and should not be able to) access that particular file directly.

This is not an error, nothing is going wrong, and you can safely ignore the message. If you noticed it because of other problems you are experiencing with our extension, please contact us for support with an explanation of what those problems are.

Systemic Differences, Accept.js vs Accept Hosted

To outline the differences of the two APIs for reference of any systems that rely on specific data being set:

- When Accept.js handles a new card at checkout, it will create a customer profile and create a payment profile, then it will create a transaction from the payment profile. (Customer Profile -> Payment Profile -> Transaction)

- When Accept Hosted handles a new card at checkout, it will create a customer profile and create a payment transaction with the card information, then it will create the payment profile from the transaction. (Customer Profile -> Transaction -> Payment Profile)

Although the Accept Hosted form is attached to a customer profile, their systems don't attach the transaction to that profile if the payment profile is created after. This entire step-by-step process happens on their server, which means we have no control over their order of operations or how they handle attaching the transactions to their profiles. For now, this is the expected behavior for this new API.

Frequently Asked Questions & Troubleshooting

Please search our solution directory for the latest answers to common questions and issues:

https://paradoxlabs.freshdesk.com/support/solutions

If your question is not answered there, open a support ticket and we'll help you out.

Does this extension support 3D Secure 2.0?

Short version: No, due to API limitations.

Although Authorize.net does support 3D Secure through third party service providers (Cardinal Commerce), Authorize.net itself does not support SCA (Strong Customer Authentication). As a result, even if you implement 3D Secure with Authorize.net, that will not put you in compliance with PSD2.

Unless you have a business presence in EU or UK, the regulation won't affect you or your customers. Read more

If you are affected, we recommend changing to Authorize.net's sister company CyberSource, which does support PSD2. All of your Authorize.net stored cards can be transferred seamlessly to our CyberSource extension. Please contact us and we can help you through the transition.

How does this payment method handle currency?

Authorize.Net only supports one currency per Authorize.net account. Magento supports currency conversion, but all transactions are processed in Magento's configured 'base currency'. The base currency and product prices can be set either globally or on a per-website basis.

This means two things:

1. Your Magento base currency *must* match your Authorize.net account currency. These are configured outside of our payment extension settings. If they do not match, all of your transactions amounts will be incorrect.
2. It is possible to run multiple base currencies (setting prices explicitly, rather than being automatically converted), but each currency must have its own website and its own Authorize.Net account.

Why are my API Login ID and Transaction Key invalid?

You may be trying to use a live Authorize.net account with the extension set to sandbox mode, or vice versa.

In order to test CIM payment processing, you need to sign up for a free developer account at Authorize.net. (The account type must be 'card not present.') After registering, you will be given an API Login ID and Transaction Key. Save these, then copy them into the Magento configuration at **Admin > Stores > Configuration > Sales > Payment Methods > Authorize.Net CIM**. Also set 'Account Type' to 'Sandbox Account', then save.

To handle to live payment processing with CIM, enter your real Authorize.net account details (API Login ID and Transaction Key) and set 'Account Type' to 'Live Merchant Account'.

Make sure that test mode is **not** enabled in your account settings at Authorize.net, and that CIM is enabled. CIM **will not work** in test mode.

How do I do an online refund from Magento?

In order to process an 'online' refund through Authorize.Net, you have to go to the **invoice** you want to refund, and click the 'Credit Memo' button from there.

If you've done that correctly, at the bottom of the page you should see a button that says 'Refund'.

As of version 5.2, clicking 'Credit Memo' from the order will automatically load the most recent invoice, so the online 'Refund' option is available. If you still see only a 'Refund Offline' button, no invoice was loaded — open the specific invoice you want to refund and click 'Credit Memo' from there.

The reason for this is that the refund needs to be associated with a particular capture transaction. An order can contain any number of captures, but every capture transaction has a specific related invoice. You refund an invoice, not an order.

Error on checkout: "You cannot add more than 10 payment profiles."

This error means Authorize.Net rejected the transaction because the customer already has 10 credit cards stored on their CIM profile. This is the maximum number Authorize.Net allows. The only way to eliminate the error is to go into the Authorize.Net account and clear out some of those payment profiles for that customer. To prevent this from recurring, enable the Use multiple CIM customer profiles setting (see Advanced Settings); each new card is then stored on its own CIM profile, avoiding the 10-profile limit.

This generally occurs under the following scenarios:

1. A customer checks out repeatedly as a guest, with small variations in their billing address. Even though it may be the same card, a new payment profile is created because of those variations. Differing inputs like 'Street' vs. 'St' or 'Boulevard' vs. 'BLVD' or even differing capitalization or punctuation will cause a new payment profile to be created. We recommend having the customer create an account as opposed to checking out as a guest. Since they will have a saved billing address, that will make the many profiles error much less likely.
2. An admin is using one email to place guest orders. Since CIM profiles are recognized by customer ID and email address, that would cause all entered cards to be associated to one CIM profile. Go into Authorize.Net to remove payment profiles related to that guest email, or use distinct email addresses for guest orders to prevent further issues.
3. A customer legitimately has used 10 different cards to order within the last 120 days. If this occurs, the only option is to remove one or more of that customer's payment profiles via Authorize.Net.

Error when refunding: "The referenced transaction does not meet the criteria for issuing a credit."

Authorize.Net does not allow capture transactions to be refunded until they have 'settled' with the merchant bank and card processor. Settlement happens once a day, usually in the evening, but varies based on your account settings. If you try to refund a transaction before it has settled, you'll receive this error message. When this happens, just wait until the next day for the bank to catch up, then try again.

Note that this will only happen if you are attempting a partial refund (refunding less than the total amount of the invoice). If you are refunding the entire invoice and the transaction has not yet settled, we automatically void it instead. You won't notice anything different.

Technical / Integration Details

Architecture

The payment method code for CIM is `authnetcim`.

The payment method code for CIM ACH is `authnetcim_ach`.

`ParadoxLabs_Authnetcim` is the payment method module, built heavily on the `ParadoxLabs_TokenBase` module. TokenBase defines a variety of interfaces and architecture for handling tokenization and stored cards cleanly.

The payment method class is `\ParadoxLabs\Authnetcim\Model\Method`. This talks to Authorize.Net through `\ParadoxLabs\Authnetcim\Model\Gateway`, and stores card information in instances of `\ParadoxLabs\Authnetcim\Model\Card`. Each of these extends an equivalent abstract class in TokenBase, and implements only the details specific to the Authorize.Net API.

Card instances are stored in table `paradoxlabs_stored_card`, and referenced by quotes and orders via a `tokenbase_id` column on payment tables.

In all cases, we strongly discourage editing our code directly to modify behavior. We cannot support customizations or modified installations. Use Magento's preferences or plugins to modify behavior if necessary. If your use case isn't covered, let us know.

Custom customer attributes

- authnetcim_profile_id (deprecated)
- authnetcim_profile_version

Custom database schema

- Added table: `paradoxlabs_stored_card`
- Added column: `quote_payment.tokenbase_id`
- Added column: `sales_order_payment.tokenbase_id`

Events

- `tokenbase_before_load_payment_info` (`method`, `customer`, `transport`, `info`): Fires before preparing method-specific information for the order payment info blocks (frontend, admin, and emails). Use this to add additional information to the payment info block.
- `tokenbase_after_load_payment_info` (`method`, `customer`, `transport`, `info`): Fires before preparing method-specific information for the order payment info blocks (frontend, admin, and emails). Use this to add additional information to the payment info block, or modify what's there by default.
- `tokenbase_before_load_active_cards` (`method`, `customer`): Fires before loading a customer's available stored cards.
- `tokenbase_after_load_active_cards` (`method`, `customer`, `cards`): Fires after loading a customer's available stored cards. Use this to modify cards available to the customer or admin.

Magento API: REST and SOAP

This module supports the Magento API via standard interfaces. You can use it to create, read, update, and delete stored cards.

If you have a specific use case in mind that is not covered, please let us know.

You can generate new cards by creating an order with our payment method (code `authnetcim`), and information for a new credit card.

To place an order with a stored card, pass that card's hash in as `additional_data` -> `card_id`.

Note: The payment data you need to submit for a new card will depend on the payment form type.

- Accept Hosted:
  - Run the payment transaction using Accept Hosted, then submit the order with `additional_data`:
    - `transaction_id`
  - Credit card numbers (`cc_number`) will not be accepted.
- Accept.js:
  - Store the credit card using Accept.js. Pass the resulting token info as the following within `additional_data`:
    - `acceptjs_key`
    - `acceptjs_value`
    - `cc_last4`
    - `cc_type`
    - `cc_exp_month`
    - `cc_exp_year`
    - `cc_cid`
  - Credit card numbers (`cc_number`) will not be accepted.
- Inline form:
  - Collect the credit card info directly. Pass it in with `additional_data` fields:
    - `cc_number`
    - `cc_type`
    - `cc_exp_month`
    - `cc_exp_year`
    - `cc_cid`

Available REST API requests below. Some response data has been omitted for brevity.

Create and update (POST, PUT) requests take three objects: `card` with primary card data, `address` with address information, and `additional` for card metadata. In responses, `address` and `additional` will be nested within `card` as `address_object` and `additional_object`. This is done for technical reasons. The data formats differ, and not all fields that are returned can be set via API (EG `in_use`, `label`). This means you cannot take a card record and directly post it back to the API to update.

Integration / Admin-Authenticated API Endpoints

These API requests allow solutions acting with an admin user login, OAUTH authentication, or token-based authentication to take action on any card in the system. Data and behavior are not limited.

GET /V1/tokenbase/:cardId (get one card by ID)
Example request:

```
GET /rest/V1/tokenbase/1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "6"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "f7d085165acdfa0ea6a0b...770111",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```


GET /V1/tokenbase/search (get multiple cards, with searchCriteria)

Example request:

```
GET /rest/V1/tokenbase/search?searchCriteria[pageSize]=1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:


```
{
    "items": [
        {
            "id": 1,
            // ... other card info
        }
    ],
    "search_criteria": {
        "filter_groups": [],
        "page_size": 1
    },
    "total_count": 51
}
```


See also: Search using REST APIs (Magento DevDocs)

POST /V1/tokenbase (create card)

Example request:

```
POST /rest/V1/tokenbase HTTP/1.1
Host: {host}
```

```
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "",
        "profile_id": "1234567890",
        "payment_id": "0987654321",
        "method": "authnetcim",
        "active": "1",
        "created_at": "2017-08-03 16:31:54",
        "last_use": "2017-08-03 16:31:54",
        "expires": "2019-06-30 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "06",
        "cc_exp_year": "2019",
        "cc_last4": "0012",
        "cc_type": "DI",
        "acceptjs_key": "...",
        "acceptjs_value": "..."
    }
}
```

Example response:

```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "06"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
```

```
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

PUT /V1/tokenbase/:cardId (update card)

Example request:

```
PUT /rest/V1/tokenbase/1 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "id": 1,
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "127.0.0.1",
        "profile_id": "1234567890",
        "payment_id": "0987654321",
        "method": "authnetcim",
        "hash": "f7d085165acdfa0ea6a0b...770111",
        "active": "1",
        "created_at": "2017-08-03 16:31:54",
        "last_use": "2017-08-03 16:31:54",
        "expires": "2019-06-30 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "06",
        "cc_exp_year": "2019",
        "cc_last4": "0012",
        "cc_type": "DI"
    }
}
```

Example response:

```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "06"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
```

```
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": " f7d085165acdfa0ea6a0b...770111",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

DELETE /V1/tokenbase/:cardId (delete card by ID)
Example request:

```
DELETE /rest/V1/tokenbase/95 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
true
```

Customer Authenticated API Endpoints

These API requests allow authenticated frontend customers to manage their stored cards. This is intended for headless implementations or app integration where card management needs to be exposed outside of Magento's standard frontend.

Customers will only be able to access and manipulate active cards assigned to their specific customer ID.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

GET /V1/tokenbase/mine/:cardHash (get one card by hash)
Example request:

```
GET /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
```

```
        "cc_exp_year": "2019",
        "cc_exp_month": "6"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

GET /V1/tokenbase/mine/search (get multiple cards, with searchCriteria)

Example request:

```
GET /rest/V1/tokenbase/mine/search?searchCriteria[pageSize]=3 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "items": [
        {
            "id": 1,
            // ... other card info
        }
    ],
    "search_criteria": {
        "filter_groups": [],
        "page_size": 3
    },
    "total_count": 5
}
```

See also: Search using REST APIs (Magento DevDocs)

POST /V1/tokenbase/mine (create card)

Example request:

```
POST /rest/V1/tokenbase/mine HTTP/1.1
Host: {host}
Content-Type: application/json
Authorization: Bearer {api_key}

{
    "card": {
        "customer_email": "email@example.com",
        "customer_id": 1,
```

```
        "customer_ip": "127.0.0.1",
        "profile_id": "1234567890",
        "payment_id": "0987654321",
        "method": "authnetcim",
        "active": "1"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "06",
        "cc_exp_year": "2019",
        "cc_last4": "0012",
        "cc_type": "DI",
        "acceptjs_key": "...",
        "acceptjs_value": "..."
    }
}
```


Example response:


```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "06"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

PUT /V1/tokenbase/mine/:cardHash (update card by hash)

Example request:

```
PUT /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "card": {
        "id": 1,
        "customer_email": "email@example.com",
        "customer_id": 1,
        "customer_ip": "127.0.0.1",
        "profile_id": "1234567890",
        "payment_id": "0987654321",
        "method": "authnetcim",
        "hash": "50b8e326b012e793957215c0361afc4b52434b26",
        "active": "1",
        "expires": "2019-06-30 23:59:59"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "06",
        "cc_exp_year": "2019",
        "cc_last4": "0012",
        "cc_type": "DI"
    }
}
```

Example response:

```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "06"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
```

```
    "customer_email": "email@example.com",
    "customer_id": 1,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```


DELETE /V1/tokenbase/mine/:cardHash (delete card by hash)

Example request:

```
DELETE /rest/V1/tokenbase/mine/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
True
```

Guest API Endpoints

These API requests allow unauthenticated frontend guest users to add and fetch an individual stored card. This is intended for headless implementations or app integration where card management needs to be exposed outside of Magento's standard frontend. Guests are not able to list, edit, delete, or reuse stored cards, so no API requests are exposed for those actions.

Guests will only be able to access and manipulate active cards, by hash, not assigned to any customer ID.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

GET /V1/tokenbase/guest/:cardHash (get one card by hash)

Example request:

```
GET /rest/V1/tokenbase/guest/50b8e326b012e793957215c0361afc4b52434b26 HTTP/1.1
Host: {host}
```

Example response:


```
{
    "id": 1,
    "in_use": true,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
        "cc_exp_year": "2019",
        "cc_exp_month": "6"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
```

```
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe"
    },
    "customer_email": "email@example.com",
    "customer_id": 0,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "50b8e326b012e793957215c0361afc4b52434b26",
    "active": "1",
    "created_at": "2017-08-03 16:31:54",
    "updated_at": "2017-09-20 14:24:14",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

POST /V1/tokenbase/guest (create card)

Example request:

```
POST /rest/V1/tokenbase/guest HTTP/1.1
Host: {host}
Content-Type: application/json

{
    "card": {
        "customer_email": "email@example.com",
        "customer_id": 0,
        "customer_ip": "127.0.0.1",
        "profile_id": "1234567890",
        "payment_id": "0987654321",
        "method": "authnetcim",
        "active": "1"
    },
    "address": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
        "vat_id": ""
    },
    "additional": {
        "cc_exp_month": "06",
        "cc_exp_year": "2019",
        "cc_last4": "0012",
        "cc_type": "DI",
        "acceptjs_key": "...",
        "acceptjs_value": "..."
    }
}
```

Example response:

```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "DI",
        "cc_last4": "0012",
```

```
        "cc_exp_year": "2019",
        "cc_exp_month": "06"
    },
    "address_object": {
        "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
        },
        "region_id": 51,
        "country_id": "US",
        "street": [
            "123 Test Ln."
        ],
        "company": "",
        "telephone": "111-111-1111",
        "postcode": "17603",
        "city": "Lancaster",
        "firstname": "John",
        "lastname": "Doe",
    },
    "customer_email": "email@example.com",
    "customer_id": 0,
    "customer_ip": "127.0.0.1",
    "profile_id": "1234567890",
    "payment_id": "0987654321",
    "method": "authnetcim",
    "hash": "9b83d4683f3d3...2309ccd65b",
    "active": "1",
    "created_at": "2017-09-25 17:41:21",
    "updated_at": "2017-09-25 17:41:21",
    "last_use": "2017-08-03 16:31:54",
    "expires": "2019-06-30 23:59:59",
    "label": "Discover XXXX-0012"
}
```

Magento API: GraphQL

This extension supports Magento's GraphQL API for checkout and all customer card management. This is intended for PWA and headless implementations where card management needs to be exposed outside of Magento's standard frontend.

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Sales > Checkout > ParadoxLabs Payment Module Settings > Enable public API**. Only enable this if you use them.

We recommend using GraphiQL or the Chrome ChromeiQL browser extension for browsing your store's GraphQL schema and testing API requests.

General Usage

Customers will only be able to access and manipulate active cards assigned to their specific customer ID.

Guests will only be able to access and manipulate active cards, by hash, not assigned to any customer ID.

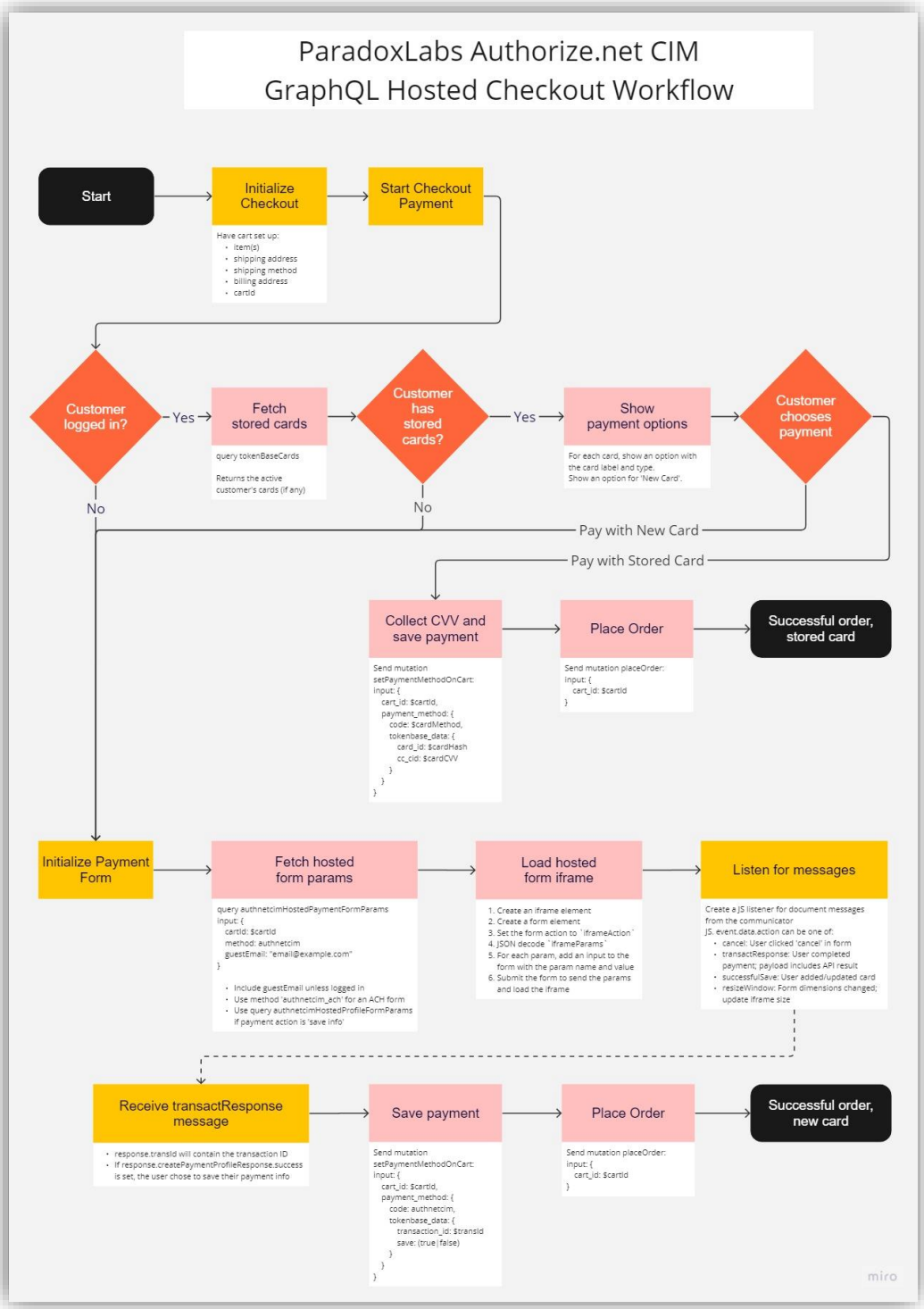
When placing an order, you will need to set the payment data using `tokenbase_data` when you call the `setPaymentMethodOnCart` mutation. See the Place an order example below.

Note: The payment data you need to submit for a new card will depend on the payment form type.

- Accept Hosted:
  - Run the payment transaction using Accept Hosted. `tokenbase_data` should include:
    - `transaction_id`
  - Credit card numbers (`cc_number`) will not be accepted.
- Accept.js:
  - Store the credit card using Accept.js. `tokenbase_data` should include:
    - `acceptjs_key`
    - `acceptjs_value`
    - `cc_last4`
    - `cc_type`
    - `cc_exp_month`
    - `cc_exp_year`
    - `cc_cid`
  - Credit card numbers (`cc_number`) will not be accepted.
- Inline form:
  - Strongly discouraged. Use a hosted form or tokenization (above) for security if at all possible.
  - Collect the credit card info directly. `tokenbase_data` should include:
    - `cc_number`
    - `cc_type`
    - `cc_exp_month`
    - `cc_exp_year`
    - `cc_cid`

Checkout Payment Flow with the Accept Hosted form

This flowchart explains the process and GraphQL steps for running checkout with the Accept Hosted payment form. See the chart in full detail at: https://miro.com/app/board/uXjVNVHzb1Q=/?share_link_id=113819350883



Queries

authnetcimHostedPaymentFormParams(input: TokenBaseAuthnetcimHostedPaymentFormInput!): [TokenBaseAuthnetcimHostedParams]

Get an Accept Hosted form URL and parameters for charging an order. If you have the payment form set to 'Accept Hosted', this gives the data necessary to create the hosted payment iframe for collecting credit card data during checkout. Takes a cart ID and payment method code; returns a form action URL and inputs.

authnetcimHostedProfileFormParams(input: TokenBaseAuthnetcimHostedProfileFormInput!): [TokenBaseAuthnetcimHostedParams]

Get an Accept Customer form URL and parameters for collecting payment data. This allows card creation and updating when the payment form is set to 'Accept Hosted'. It differs from the PaymentForm above in that it will save a card payment profile, but not initiate any payment transaction.

tokenBaseCards(hash: String): [TokenBaseCard]

Get the current customer's stored card(s), if any. Takes a card hash (optional); returns one or more `TokenBaseCard` records. If no hash is given, will return all active cards belonging to the customer.

tokenBaseCheckoutConfig(method: String!): TokenBaseCheckoutConfig

Get checkout configuration for the given TokenBase payment method. Takes a TokenBase payment method code, such as `authnetcim`; returns a `TokenBaseCheckoutConfig`. This returns all data necessary to render and handle the client-side checkout form. Values mirror what is passed to Magento's standard frontend checkout.

Mutations

authnetcimSyncHostedForm(input: TokenBaseAuthnetcimHostedProfileFormInput!): TokenBaseAuthnetcimHostedCardResponse

Run after the user completes an Accept Customer hosted profile form, to import their card from Authorize.net into Magento. Takes one or more identifiers (cartId, cardId, iframeSessionId) to tie it back to the profile form that was submitted, along with a payment method code and source. Returns a card including `TokenBaseAuthnetcimHostedCard` for display or checkout purposes.

createTokenBaseCard(input: TokenBaseCardCreateInput!): TokenBaseCard

Create a new stored card. Takes `TokenBaseCardCreateInput`, returns the new stored `TokenBaseCard` if successful.

deleteTokenBaseCard(hash: String!): Boolean

Delete a stored card. Takes a card hash; returns true if successful.

updateTokenBaseCard(input: TokenBaseCardUpdateInput!): TokenBaseCard

Update an existing stored card. Takes `TokenBaseCardUpdateInput`; returns the updated `TokenBaseCard` if successful.

Data Types

TokenBaseAuthnetcimHostedPaymentFormInput

Parameters to initialize a Hosted payment iframe. Takes a cart ID, payment method code, and `guestEmail` (for guest checkouts only).

```
input TokenBaseAuthnetcimHostedPaymentFormInput {
        cartId: String
        method: TokenBaseAuthnetcimHostedMethod!
        guestEmail: String
}
```

TokenBaseAuthnetcimHostedProfileFormInput

Parameters to initialize a Hosted profile iframe. Takes an identifier (one or more of cart ID, card ID hash, or `iframeSessionId` from a previous response), payment method code, source (`paymentinfo` or `checkout`) and `guestEmail` (for guest checkouts only).

```
input TokenBaseAuthnetcimHostedProfileFormInput {
        cartId: String
        cardId: String
        iframeSessionId: String
        method: TokenBaseAuthnetcimHostedMethod!
        source: TokenBaseAuthnetcimHostedSource!
        guestEmail: String
}
```

TokenBaseAuthnetcimHostedParams

Response for an Accept Hosted payment form or profile form query. JSON decode the `iframeParams`. POST those params to the `iframeAction` URL, targeting an iframe, to create the hosted payment form. Use the `iframeSessionId` on follow up GraphQL requests as needed.

```
type TokenBaseAuthnetcimHostedParams {
        iframeSessionId: String
        iframeAction: String
        iframeParams: String
}
```

TokenBaseAuthnetcimHostedCardResponse

Response after completing an Accept Customer profile form, and then running `authnetcimSyncHostedForm` to sync the card to Magento. Includes `TokenBaseAuthnetcimHostedCard` with the new card details (if any).

```
type TokenBaseAuthnetcimHostedCardResponse {
        card: TokenBaseAuthnetcimHostedCard
}
```

TokenBaseAuthnetcimHostedCard

Metadata for a newly stored Accept Customer payment profile.

```
type TokenBaseAuthnetcimHostedCard {
        id: String
        label: String
        selected: Boolean
        new: Boolean
        type: String
        cc_bin: String
        cc_last4: String
}
```

TokenBaseCard

A stored payment account/credit card.

```
type TokenBaseCard {
        hash: String                                    Card identifier hash
        address: CustomerAddress                        Card billing address
        customer_email: String                          Customer email
        customer_id: Int                                Customer ID
        customer_ip: String                             Created-by IP
        profile_id: String                              Card gateway profile ID
        payment_id: String                              Card gateway payment ID
        method: String                                  Payment method code
        active: Boolean                                 Is card active
        created_at: String                              Created-at date
        updated_at: String                              Last updated date
        last_use: String                                Last used date
        expires: String                                 Expiration date
        label: String                                   Card label
        additional: TokenBaseCardAdditional             Card payment data
```

```
}
```

TokenBaseCardAdditional

Details and metadata for a stored CC/ACH.

```
type TokenBaseCardAdditional {
        cc_type: String                                 CC Type
        cc_owner: String                                CC Owner
        cc_bin: String                                  CC Bin (CC First-6)
        cc_last4: String                                CC Last-4
        cc_exp_year: String                             CC Expiration Year
        cc_exp_month: String                            CC Expiration Month
        echeck_account_name: String                     ACH Account Name
        echeck_bank_name: String                        ACH Bank Name
        echeck_account_type: TokenBaseEcheckAccountType ACH Account type
        echeck_routing_number_last4: String             ACH Routing Number Last-4
        echeck_account_number_last4: String             ACH Account Number Last-4
}
```

TokenBaseCheckoutConfig

Checkout configuration for a TokenBase payment method.

```
type TokenBaseCheckoutConfig {
        method: String                                  Payment method code
        useVault: Boolean                               Are stored cards enabled?
        canSaveCard: Boolean                            Can cards be saved?
        forceSaveCard: Boolean                          Is card saving forced?
        defaultSaveCard: Boolean                        Hash of the default card to select
        isCcDetectionEnabled: Boolean                   Is CC type detection enabled?
        logoImage: String                               Payment logo image URL (if enabled)
        requireCcv: Boolean                             Is CVV required for stored cards?
        sandbox: Boolean                                Is the payment gateway in sandbox mode?
        canStoreBin: Boolean                            Is CC BIN (CC first6) storage enabled?
        availableTypes: [TokenBaseKeyValue]             Available CC types
        months: [TokenBaseKeyValue]                     Available CC Exp Months
        years: [TokenBaseKeyValue]                      Available CC Exp Years
        hasVerification: Boolean                        Is CVV enabled?
        cvvImageUrl: String                             CVV helper image URL
}
```

TokenBaseKeyValue

Container for generic key/value data.

```
type TokenBaseKeyValue {
        key: String                                     Generic key
        value: String                                   Generic value
}
```

TokenBaseCardUpdateInput

Input for updating a stored card.

```
input TokenBaseCardUpdateInput {
        hash: String!                                   Card identifier hash to update (required)
        address: CustomerAddressInput                   Card billing address
        customer_email: String                          Customer email
        customer_ip: String                             Created-by IP
        method: String                                  Payment method code
        active: Boolean                                 Is card active
        expires: String                                 Card expiration date (YYYY-MM-DD 23:59:59)
        additional: TokenBaseCardPaymentInput           Card payment data
}
```

TokenBaseCardCreateInput

Input for creating a stored card.

```
input TokenBaseCardCreateInput {
        address: CustomerAddressInput                   Card billing address
        customer_email: String!                         Customer email (required)
```

```
        customer_ip: String                                 Created-by IP
        method: String!                                     Payment method code (required)
        active: Boolean                                     Is card active
        expires: String                                     Card expiration date (YYYY-MM-DD 23:59:59)
        additional: TokenBaseCardPaymentInput               Card payment data
}
```

TokenBaseCardPaymentInput

Payment data for a stored card. Note, the specific fields that are relevant depend on the payment method. This data structure is also used for adding payment data to the cart during checkout.

```
input TokenBaseCardPaymentInput {
        cc_type: String                                     CC Type
        cc_owner: String                                    CC Owner
        cc_bin: String                                      CC Bin (CC First-6)
        cc_last4: String                                    CC Last-4
        cc_number: String                                   CC Number
        cc_cid: String                                      CC CVV
        cc_exp_year: String                                 CC Expiration Year
        cc_exp_month: String                                CC Expiration Month
        echeck_account_name: String                         ACH Account Name
        echeck_bank_name: String                            ACH Bank Name
        echeck_account_type: TokenBaseEcheckAccountType     ACH Account Type
        echeck_routing_number: String                       ACH Routing Number
        echeck_account_number: String                       ACH Account Number
        acceptjs_key: String                                Accept.js Key
        acceptjs_key: Value                                 Accept.js Value
        save: Boolean                                       Save the card for later use? (checkout only)
        card_id: String                                     Card identifier hash to use (checkout only)
        transaction_id: String                              Transaction ID (Accept Hosted checkout only)
}
```

GraphQL Query Examples

Some response data has been omitted for brevity.

Fetch card by ID

Example request:

```
{
  tokenBaseCards(hash:"88bb7dc06faad55c77177446ed83047811234008") {
    label,
    expires,
    hash,
    customer_email,
    customer_id,
    profile_id,
    payment_id,
    method,
    active,
    created_at,
    updated_at,
    last_use,
    address {
      region {
        region_code,
        region,
        region_id
      },
      region_id,
      country_id,
      street,
      company,
      telephone,
      postcode,
      city,
      firstname,
      lastname
    },
    additional {
      cc_type,
      cc_last4,
      cc_exp_year,
      cc_exp_month
```

```
    }
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCards": [
      {
        "label": "Discover XXXX-0012",
        "expires": "2021-03-31 23:59:59",
        "hash": "88bb7dc06faad55c77177446ed83047811234008",
        "customer_email": "roni_cost@example.com",
        "customer_id": 1,
        "profile_id": "1200144368",
        "payment_id": "1506360102",
        "method": "authnetcim",
        "active": true,
        "created_at": "2019-03-11 16:12:52",
        "updated_at": "2019-04-04 18:01:39",
        "last_use": "2019-04-04 18:01:39",
        "address": {
          "region": {
            "region_code": "MI",
            "region": "Michigan",
            "region_id": 33
          },
          "region_id": 33,
          "country_id": "US",
          "street": [
            "6146 Honey Bluff Parkway"
          ],
          "company": "",
          "telephone": "(555) 229-3326",
          "postcode": "49628-7978",
          "city": "Calder",
          "firstname": "Veronica",
          "lastname": "Costello"
        },
        "additional": {
          "cc_type": "DI",
          "cc_last4": "0012",
          "cc_exp_year": "2021",
          "cc_exp_month": "3"
        }
      }
    ]
  }
}
```

Fetch checkout config

Example request:

```
{
  tokenBaseCheckoutConfig(method:"authnetcim") {
    method,
    useVault,
    canSaveCard,
    forceSaveCard,
    defaultSaveCard,
    isCcDetectionEnabled,
    logoImage,
    requireCcv,
    sandbox,
    canStoreBin,
    availableTypes {key, value},
    months {key, value},
    years {key, value},
    hasVerification,
    cvvImageUrl,
    apiLoginId,
    clientKey
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCheckoutConfig": {
      "method": "authnetcim",
      "useVault": true,
      "canSaveCard": true,
      "forceSaveCard": false,
      "defaultSaveCard": true,
      "isCcDetectionEnabled": true,
      "logoImage": "https://.../images/logo.png",
      "requireCcv": false,
      "sandbox": true,
      "canStoreBin": false,
      "availableTypes": [
        {
          "key": "AE",
          "value": "American Express"
        },
        ...
      ],
      "months": [
        {
          "key": "1",
          "value": "01 - January"
        },
        ...
      ],
      "years": [
        {
          "key": "2019",
          "value": "2019"
        },
        ...
      ],
      "hasVerification": true,
      "cvvImageUrl": "https://.../Magento_Checkout/cvv.png",
      "apiLoginId": "ABCDEFGHI",
      "clientKey": "9777muF7X2uQc3fv5DucBx…"
    }
  }
}
```

Create card

Example request:

```
mutation {
  createTokenBaseCard(
    input: {
      expires: "2022-12-31 23:59:59",
      customer_ip: "127.0.0.1",
      customer_email: "roni_cost@example.com",
      method: "authnetcim",
      active: true,
      address: {
        region: {
          region_code: "PA",
          region: "Pennsylvania",
          region_id: 51
        },
        country_id: US,
        street: [
          "123 Test St.",
          "Apt 9"
        ],
        company: "",
        telephone: "111-111-1111",
        postcode: "12345",
        city: "Testcity",
        firstname: "John",
        lastname: "Doe"
      },
      additional: {
        cc_type: "VI",
        cc_last4: "0027",
        cc_exp_year: "2022",
```

```
        cc_exp_month: "12",
        cc_cid: "123",
        acceptjs_key: "COMMON.ACCEPT.INAPP.PAYMENT",
        acceptjs_value: "eyJjb2RlIjoiNT…"
      }
    }
  ) {
    label,
    expires,
    hash,
    customer_email,
    customer_id,
    customer_ip,
    profile_id,
    payment_id,
    method,
    active,
    created_at,
    updated_at,
    last_use,
    address {
      region {
        region_code,
        region,
        region_id
      },
      region_id,
      country_id,
      street,
      company,
      telephone,
      postcode,
      city,
      firstname,
      lastname
    },
    additional {
      cc_type,
      cc_last4,
      cc_exp_year,
      cc_exp_month
    }
  }
}
```

Example response:

```
{
  "data": {
    "createTokenBaseCard": {
      "label": "Visa XXXX-0027",
      "expires": "2021-03-31 23:59:59",
      "hash": "88bb7dc06faad55c77177446ed83047811234008",
      "customer_email": "roni_cost@example.com",
      "customer_id": 1,
      "profile_id": "1200144368",
      "payment_id": "1506360102",
      "method": "authnetcim",
      "active": true,
      "created_at": "2019-03-11 16:12:52",
      "updated_at": "2019-04-04 18:01:39",
      "last_use": "2019-04-04 18:01:39",
      "address": {
        "region": {
          "region_code": "MI",
          "region": "Michigan",
          "region_id": 33
        },
        "region_id": 33,
        "country_id": "US",
        "street": [
          "6146 Honey Bluff Parkway"
        ],
        "company": "",
        "telephone": "(555) 229-3326",
        "postcode": "49628-7978",
        "city": "Calder",
        "firstname": "Veronica",
```

```
        "lastname": "Costello"
      },
      "additional": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2022",
        "cc_exp_month": "12"
      }
    }
  }
}
```

Delete card

Example request:

```
mutation {
  deleteTokenBaseCard(hash:"88bb7dc06faad55c77177446ed83047811234008")
}
```

Example response:

```
{
  "data": {
    "deleteTokenBaseCard": true
  }
}
```

Fetch Accept Hosted iframe parameters

Example request:

```
query {
  authnetcimHostedPaymentFormParams(input: {
    cartId: “kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1”
    method: authnetcim
    guestEmail: "email@example.com"
  }) {
    iframeAction
    iframeParams
  }
}
```

Example response:

```
{
  "data": {
    "authnetcimHostedPaymentFormParams": {
      "iframeAction": "https://test.authorize.net/payment/payment",
      "iframeParams":
"{\"token\":\"3+mqYlzWjwiDtOlq8WDu4wu3SU1zKNyRkI5ibPOtJB1HoALm36i2LU2W7VkRNOm6LuRICCeQ794FoZqiTecnM8SQRRnw4M\\
/2scZL4NN5R5ioMFmoL+1blOMC2fHy6OAIG13+j2J2MUz4tZzJMa9C\\/EOEebrSuoD6+SXwmimM8SC2GCJkuQtrPp48RIy2bLTfQtzvGM1QBr
mM73LIV3Qzj7pUpUZ3V7aWIXRRO3MOh96pZOACBA1RAB5+tJxiXvJ7pje3ezjYnDC+clThHFpgRy2hgL11eqS\\/wtpDbaauWixHKKOecUSkrv
9x7RUCAtv6hvgN37eJyvWEP4hkHOuw\\/Ex5tlt3we...d1XHx\\/S8RajLnsoTzM95lNmIh.69f8DJBmx\"}"
    }
  }
}
```

Place an order

Example request:

```
mutation {
  setPaymentMethodOnCart(
    input: {
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
      payment_method: {
        code: "authnetcim",
        tokenbase_data: {
          save: true
          # ACCEPT.JS PARAMS:
          cc_type: "DI",
```

```
          cc_last4: "1111",
          cc_exp_year: "2022",
          cc_exp_month: "08",
          cc_cid: "123",
          acceptjs_key: "COMMON.ACCEPT.INAPP.PAYMENT",
          acceptjs_value: "eyJjb2RlIjoiNT…"
          # ACCEPT HOSTED PARAMS:
          transaction_id: “80007594831”
        }
      }
    }
  ) {
    cart {
      selected_payment_method {
        code
      }
    }
  }
  placeOrder(
    input: {
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    }
  ) {
    order {
      order_id
    }
  }
}
```

Example response:

```
{
  "data": {
    "setPaymentMethodOnCart": {
      "cart": {
        "selected_payment_method": {
          "code": "authnetcim"
        }
      }
    },
    "placeOrder": {
      "order": {
        "order_id": "000000255"
      }
    }
  }
}
```

Support

If you have any questions not covered by this document, or something isn’t working right, please open a ticket in our support system: support.paradoxlabs.com

Support Policy: https://store.paradoxlabs.com/support.html

License and Terms of Use: https://store.paradoxlabs.com/license.html