Authorize.Net CIM: User Manual

Version 4.3 – For Magento® 2.2+ – Updated 2021-03-29

Table of Contents

Installation

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

If you purchased from Magento Marketplace

NOTE: You **will not** be able to install by downloading the extension files from Marketplace.
The Marketplace download does not include all of the necessary files. You must install using either the Web Setup Wizard or Composer, with the following directions.

Step 1: Install

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

If you encounter any problems during this process, please contact [Magento Marketplace Support](#).

Note, installing this extension requires familiarity with your server's command line. Ensure your server has composer set up and linked to your Magento Marketplace account (including repository https://repo.magento.com). Then in SSH, from your site root, run the following commands:

```
composer require paradoxlabs/authnetcim
php bin/magento module:enable -c ParadoxLabs_TokenBase ParadoxLabs_Authnetcim
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
```

These commands should load and install the extension packages from the Marketplace repository.

Composer installation is only available for Marketplace purchases.

Step 2: Configure

See the configuration section below.

If you purchased from store.paradoxlabs.com

NOTE: This file upload installation applies **only** to purchases from the ParadoxLabs Store. Marketplace purchases must follow the Marketplace 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 module:enable -c ParadoxLabs_TokenBase ParadoxLabs_Authnetcim
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
```

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 purchased from Magento 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
```

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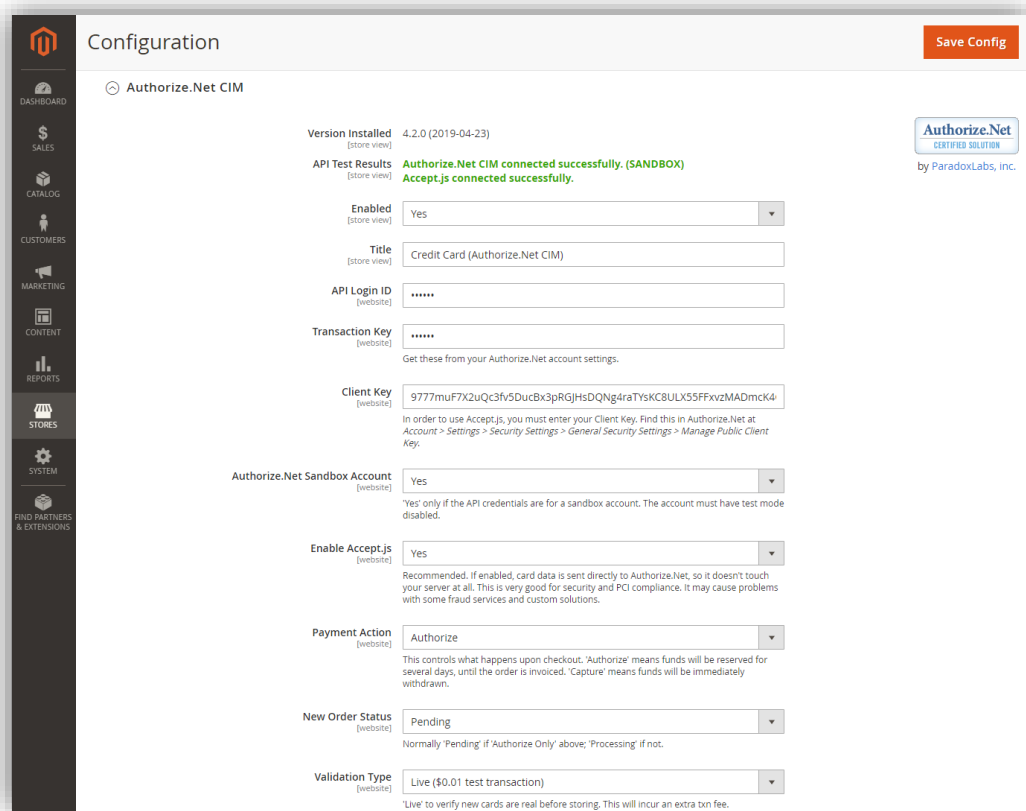
```
php bin/magento setup:di:compile
php bin/magento setup:static-content:deploy
```

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 > Settings > 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 tells you the version of our extension currently installed on your website. Please include this in any support requests.
- **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 we cannot connect to Authorize.Net, or your API Login ID or Transaction Key is incorrect, or your Authorize.Net account does not have the CIM service enabled, this will tell you with a red message. Correct the error, then reload the page and it should show 'Authorize.Net CIM connected successfully.'
- **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 status pages.
- **API Login ID**: 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 **Tools > Security Settings > General Security Settings > API Login ID and Transaction Key**. 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 you're 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.
- **Client Key**: In order to use Accept.js for enhanced security, 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**.
- **Authorize.Net Sandbox Account**: Choose 'Yes' if the API Login ID and Transaction Key you entered are for a sandbox account. If this value is not correct, the API Test Results will report *'Your API credentials are invalid.'* If you want to test, you must have a sandbox account (separate from your production Authorize.Net account). You can create one here: https://developer.authorize.net/hello_world/sandbox/
- **Accept.js**: If yes, and you've entered your Client Key (above), Authorize.Net's Accept.js functionality will be enabled everywhere credit card info is entered. This sends CC data straight to Authorize.Net, bypassing your server entirely. Unless it causes problems, we strongly recommend using this, for security and PCI-compliance reasons. See the full details on what Accept.js is and how it works in the section later in this document.
- **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. If Validation Type is set to 'live', it will be tested to verify it is a valid credit card in the process. No funds will be captured or held from the credit card upon checkout. Invoicing the order will perform a standalone authorize+capture transaction, but is not guaranteed to go through.
  - **Authorize**: This will authorize the order amount upon checkout, allowing for manual invoicing and capture of the funds later. 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).
  - **Authorize and Capture**: This will capture all funds immediately when an order is placed.

  Payment processors strongly recommend not capturing funds unless/until you are within three days of fulfilling (shipping) the order
- **New Order Status**: Set this to your desired initial status for orders paid via Authorize.Net CIM. Default Magento behavior is 'Pending' for Authorize Only, and 'Processing' for Authorize and Capture.
- **Validation Type**: Choose from the following options.
  - **Live**: This will run a $0.00 or $0.01 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 you are guaranteed all cards stored in Authorize.Net CIM (and visible on customers' accounts) are 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.
  - **Test**: In this mode, Authorize.Net will verify that the credit card number and expiration date are possible, but will not actually talk to the card processor. There is no guarantee that the CCV or billing address are correct, or that the card has any funds available.
  - **None**: In this mode, Authorize.Net will blindly store the card without any validation.

- **Show Authorize.Net Logo**: If yes, checkout will display an 'Authorize.Net' logo next to the payment form.

Payment Settings

- **Allow Credit Card Types**: Choose the CC types you want to allow on checkout.
- **Credit Card Verification**: If yes, customers will be prompted for their credit card's CCV code when entering a new card.
- **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 be given 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 120 days (the maximum refund period) after its last use. This ensures the info is available for edits, captures, and refunds, but respects the customer' 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 select it as a payment option.
- **Minimum Order Total**: This setting allows you to set a minimum order value for the payment option. 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 option. 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.

Advanced Settings

Advanced Settings

Require CCV for all transactions [website]	No ▼
	If yes, CCV code will be required even for stored cards. This will not affect recurring transactions.
Send Shipping Address [website]	Yes ▼
	If yes, shipping address will be sent with each transaction. Required for some fraud filters.
Send Line Items [website]	Yes ▼
	If yes, the items will be included with each transaction. May not be desired for some accounting integrations.
Reauthorize on Partial Invoice [website]	Yes ▼
	If yes, when you create a partial invoice, we will reauthorize any outstanding balance on the order. This helps guarantee funds, but can cause multiple holds on the card until transactions settle.
Auto-select 'save for next time' [website]	Yes ▼
	If yes, will be selected by default during checkout.
Verify SSL [website]	Yes ▼
	Strongly recommended. Do not disable unless you get SSL errors and your host is unable to fix them.

- **Require CCV for all transactions**: If yes, customers and admins will be required to enter the credit card CCV for all transactions, even with previously-stored cards.
- **Send Shipping Address**: If yes, shipping address will be included with every new transaction sent to Authorize.Net. This is required for some fraud filters. There is no performance penalty associated with this option. Shipping addresses are not stored in CIM.
- **Send Line Items**: If yes, the order items will be included with every transaction. This is required for Level-2 processing, but may not be desired for other reasons. There is no performance penalty associated with this option.
- **Reauthorize on Partial Invoice**: If yes, and you invoice part of an order, a new authorization will be created for the outstanding order balance (if any). This helps guarantee funds, but can cause multiple holds on the card until transactions settle. Any failure during reauthorization is ignored.
- **Auto-select 'save for next time'**: If yes, the 'save this card for next time' checkbox will be checked by default. If no, customers will have to explicitly select it 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 store vulnerable to MITM (man-in-the-middle) attacks.

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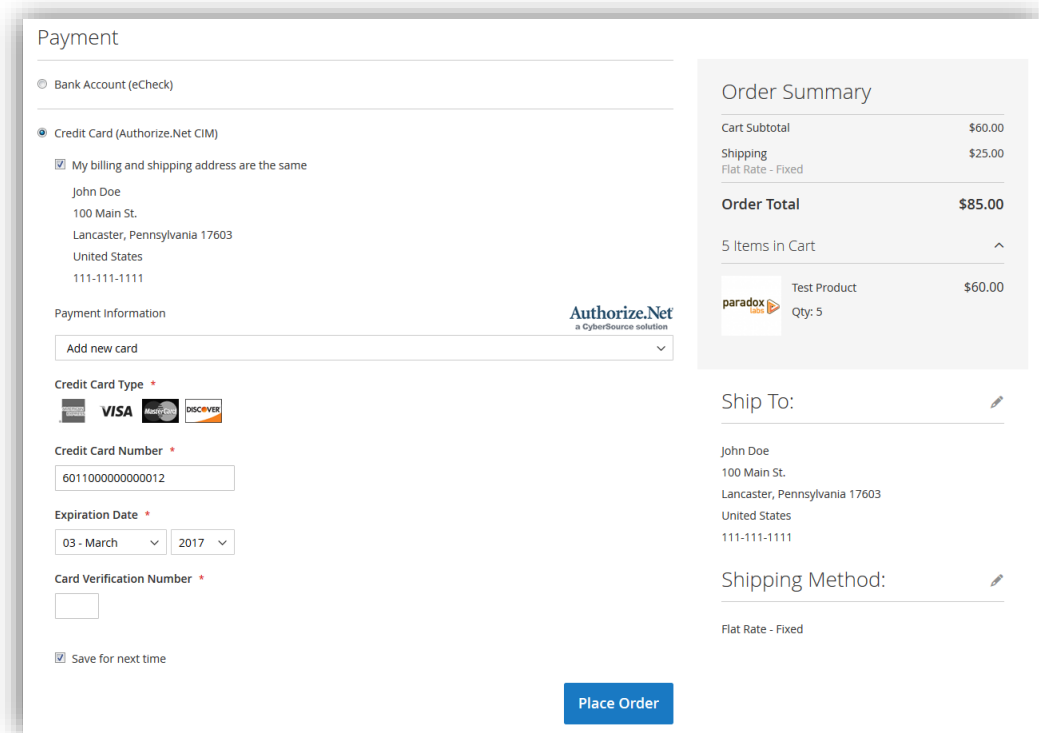
Note that there are significant differences in how ACH payments work compared to credit cards. Although all processes appear the same to Magento, when and how money is moved is quite different. We strongly recommend familiarizing yourself with the eCheck.Net FAQ: https://support.authorize.net/s/article/eCheck-Net-FAQ

Usage

There isn't much to using Authorize.Net CIM in practice: It's a standard Magento payment method, and all interfaces should be pretty self-explanatory. That being said, 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.

If a customer re-enters a card they've used before, the existing card will be detected, and the new information (expiration date, billing address, etc.) will be saved on top of it. This happens seamlessly behind the scenes.

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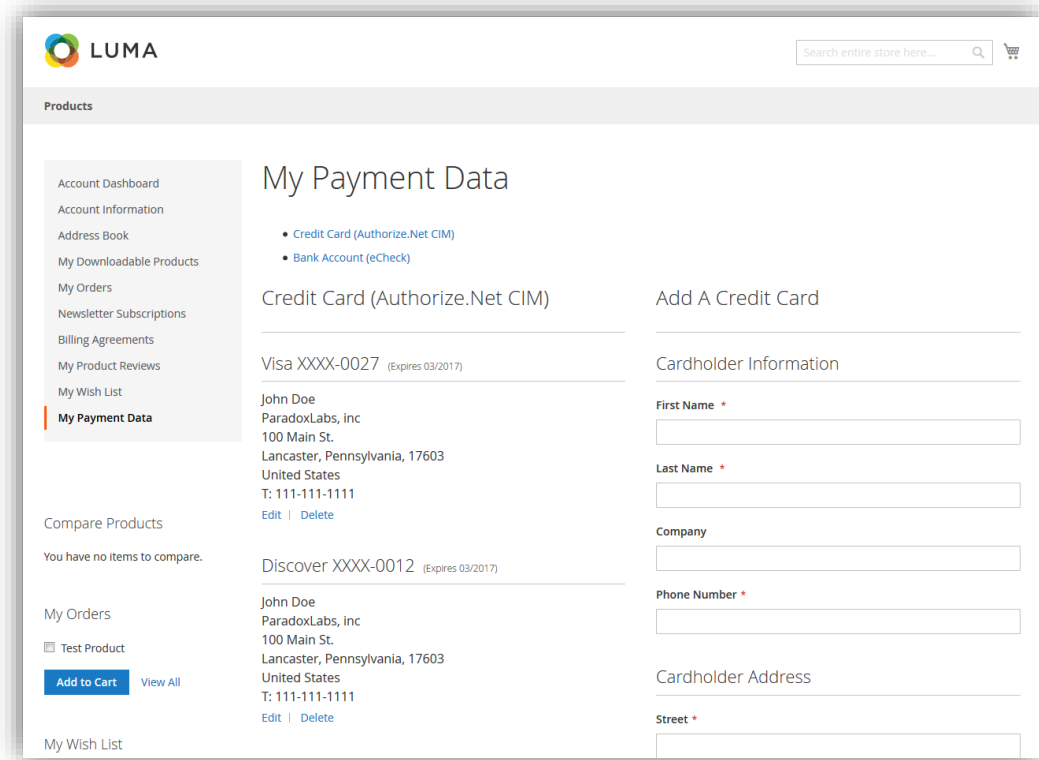
Note that cards associated with an open (uncaptured) order cannot be edited or deleted. They will display a *'Card In Use'* message in place of the buttons. As soon as all orders paid by the card are completed, the 'Edit' and 'Delete' buttons will appear.

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."* Both of these behaviors can be adjusted or disabled by internal configuration if necessary; please contact us if you have a problem.

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

As of version 4.3.0, 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/s/article/Account-Updater-FAQs

Note that there are transaction fees associated with the Account Updater service, if enabled.

Accept.js: Same Authorize.Net, even better security

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

Accept.js is Authorize.Net's modern alternative to Direct Post Method. Accept.js allows credit card information to be sent straight from your customers' browsers to Authorize.Net, without touching your web server at all. Instead, Authorize.Net gives us a one-time-use token (nonce) that refers to it. Since your web server never sees the raw credit card number, this improves your website's security, and reduces your PCI compliance exposure.
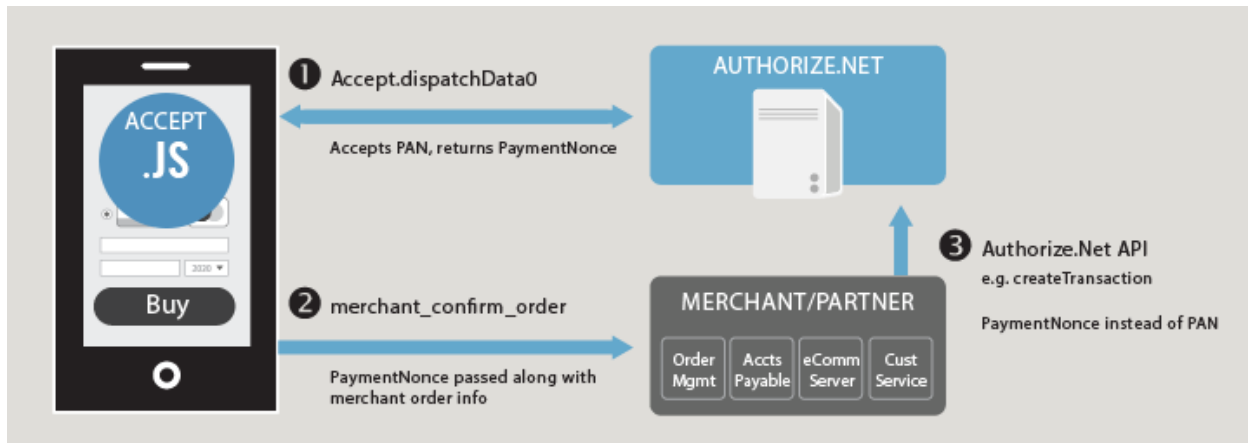
Image © 2016 Authorize.Net (used by permission)

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 validation or 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 'Accept.js' setting to 'Yes' at **Admin > Stores > Settings > Configuration > Sales > Payment Methods > Authorize.Net CIM > Accept.js**.

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* all pages having a payment form to use SSL (including your admin panel and dev sites). Also, while we've done the best we can to ensure compatibility, we cannot guarantee the additional functionality will work with every possible payment form and checkout solution. 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.

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?

Much has been made about the need for Magento merchants to support 3D Secure 2.0, due to the new legislation (PSD2) that came into effect in much of EU in September 2019. 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 implementing 3DS2 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. Please contact us and we can help you through the transition.

How do I migrate data from my site running Authorize.Net CIM for Magento 1?

To move from our extension on Magento 1.x to Magento 2, first you'll have to purchase this new version for compatibility. Then, you (or your developer, agency, or system integrator) will have to use the Magento Data Migration Tool to convert all of your data to the new site.

You will have to do some configuration of the data migration tool to include our extension's data fields. You can download a guide on the process here: Migrating Authorize.Net CIM data from Magento 1 to 2

That being said, please be aware that this is a fairly complex process, and data migration is not covered under extension support.

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 our 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 > Settings > Configuration > Sales > Payment Methods > Authorize.Net CIM**. Also set 'Sandbox Account' to 'Yes', 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 'Sandbox Account' to 'No'.

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'.

If you only have one button that says 'Refund Offline', it's because you clicked 'Credit Memo' from the order instead of from the invoice.

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.

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.

Error on installation: "Init vector must be a string of 32 bytes."

This error occurs when encrypted settings are migrated from Magento 1 to Magento 2 without the encryption being updated. You can either remove the bad values from database table `core_config_data` and re-enter them after installing, or update your migration tool configuration and remigrate. [Read more](#)

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
- 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 that raw credit card numbers (`cc_number`) will not be accepted if Accept.js is enabled. If your solution is able to implement [Accept.js](#), you can pass those details as paymentData values `acceptjs_key`, `acceptjs_value`, and `cc_last4` instead.

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

For Magento 2.3.1+, this extension supports the GraphQL API for 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.

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.

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.

Queries

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

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

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)
}
```

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
  }
}
```

Place an order

Example request:

```
mutation {
  setPaymentMethodOnCart(
    input: {
      cart_id: "kDy0EKkJmIOxa6H3ceus6MjMaSF9lao1"
      payment_method: {
        code: "authnetcim",
        tokenbase_data: {
          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…",
          save: true
        }
      }
    }
  ) {
    cart {
      selected_payment_method {
        code
      }
    }
  }
  placeOrder(
    input: {
      cart_id: "kDy0EKkJmIOxa6H3ceus6MjMaSF9lao1"
    }
  ) {
    order {
      order_id
    }
  }
}
```

Example response:

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

Split Database

This module fully supports Magento Enterprise's split database feature. No special setup should be necessary to get it working in a split-database environment. If you encounter any problems, please let us know.

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