ParadoxLabs Carat Payments: User Manual

Version 1.1 – For Magento® 2.4 – Updated 2024-06-28

Table of Contents

Installation

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

If you purchased from Adobe Commerce/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 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 repo.magento.com). Then in SSH, from your site root, run the following commands:

```
composer require paradoxlabs/carat:*
php bin/magento module:enable -c ParadoxLabs_TokenBase ParadoxLabs_Carat
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_Carat
php bin/magento setup:upgrade
```

These will enable the module, and then run the installation process.

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 the Extension

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 composer.

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
```

Connecting A New Carat Account

Before proceeding: Contact Carat to sign up for merchant account if you don't have one already. You will need to go through the account setup and activation process before you can accept real payments.

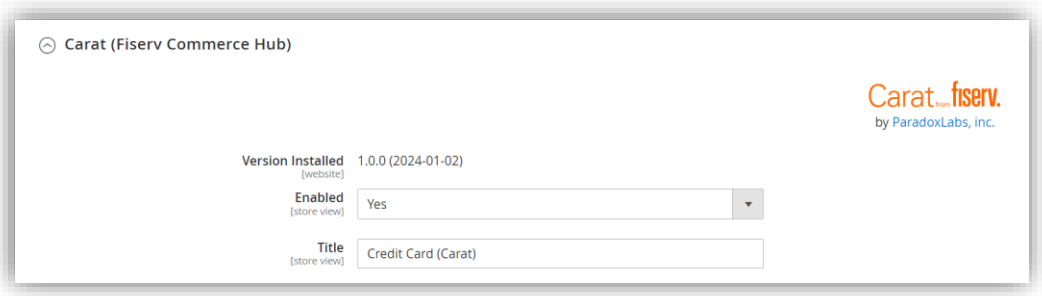
In order to use this extension, you must have a Fiserv Developer Studio account. This works in conjunction with your Carat merchant account and allows your Carat account to connect to third parties via the Fiserv Commerce Hub API.

You must have a Fiserv Developer Studio account, even for a production site.

Please create a Fiserv Developer Studio account at https://developer.fiserv.com/carat, and then create a Workspace for your Magento site.

Step 1. Finding the Magento configuration

Open your Admin Panel and go to **Admin > Stores > Settings > Configuration > Sales > Payment Methods**. Toward the bottom of the page, you'll find a 'Carat' settings section like this:



When you see this, you're at the right place. This is where you'll enter all of the API credentials, and set additional payment method configuration options. For the top section:

1. Leave 'Enabled' set to No until we're done.
2. If you'd like, change the Title while you're here.

Let's move on to API setup.

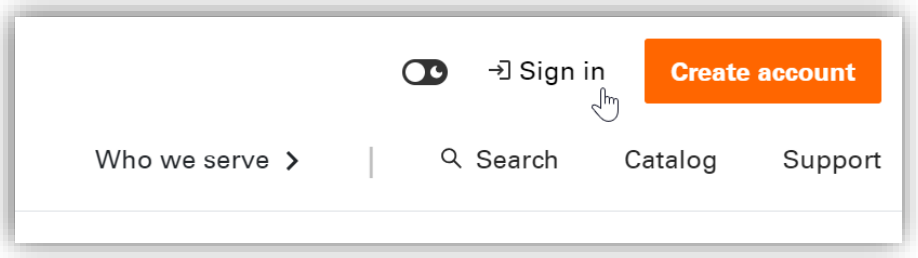
Step 2. API Setup

The next config section is to connect Magento to your Carat account. You will need the following data:

- Merchant ID (MID)
- Terminal ID
- API Key
- API Secret

To find these values, first go to the Fiserv Developer Studio at https://developer.fiserv.com.

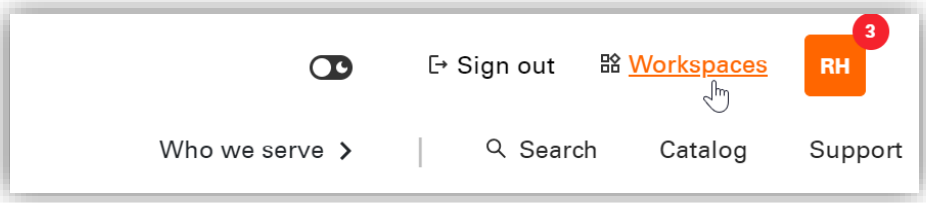
At the top right, click **Create account**, or **Sign In** if you already have an account.



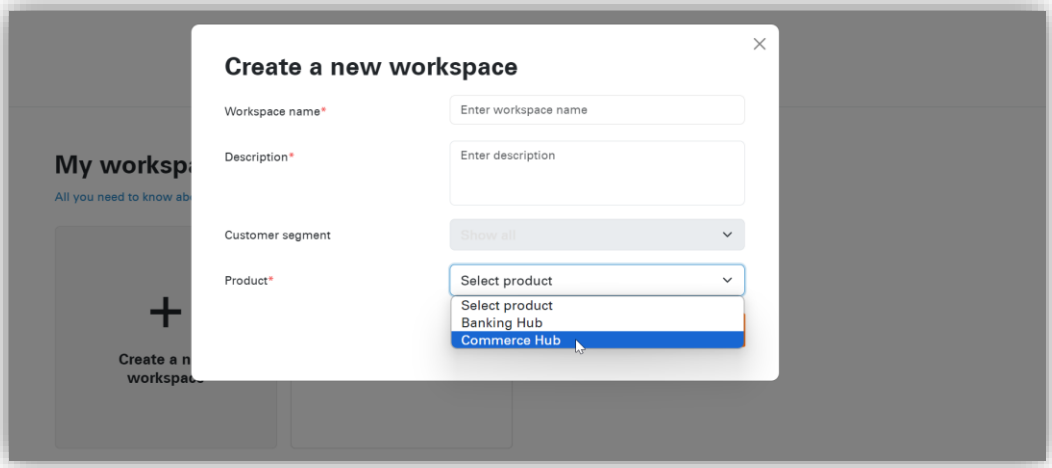
In order to use this extension, you must have a Fiserv Developer Studio account. This works in conjunction with your Carat merchant account and allows your Carat account to connect to third parties via the Fiserv Commerce Hub API. It is separate a separate account and login from your Carat merchant account.

You must create a Fiserv Developer Studio account, even for a production site, even if you already have a Carat account.

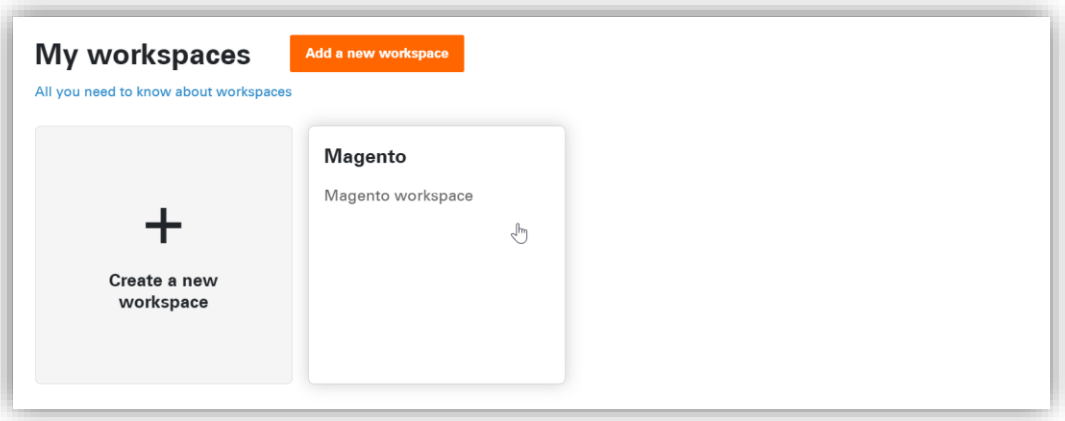
Once you have created and logged into your Developer Studio account, you will see the following in the site header. Click **Workspaces**.



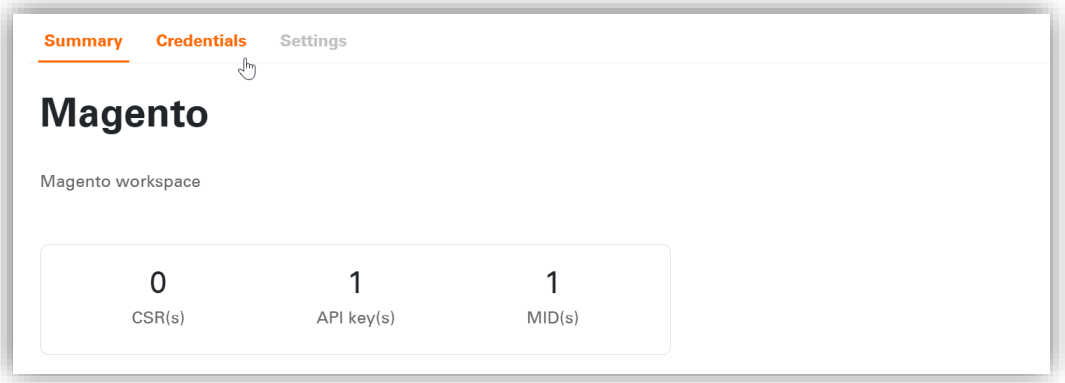
If you don't have a workspace for your website yet, Create a new workspace. Set the Product to **Commerce Hub**.



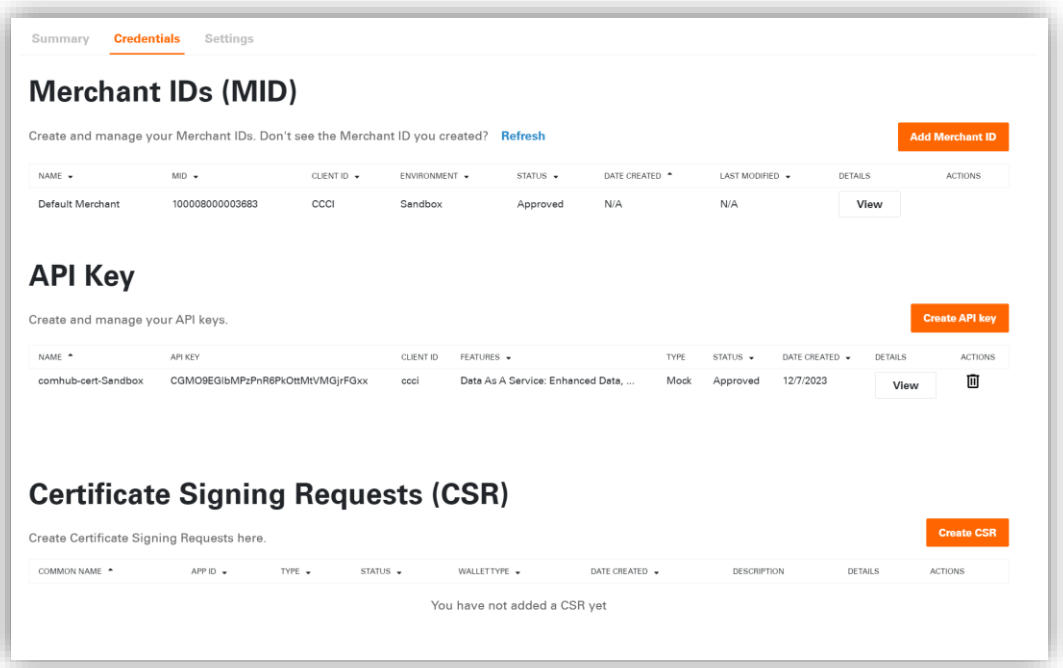
Select your workspace.



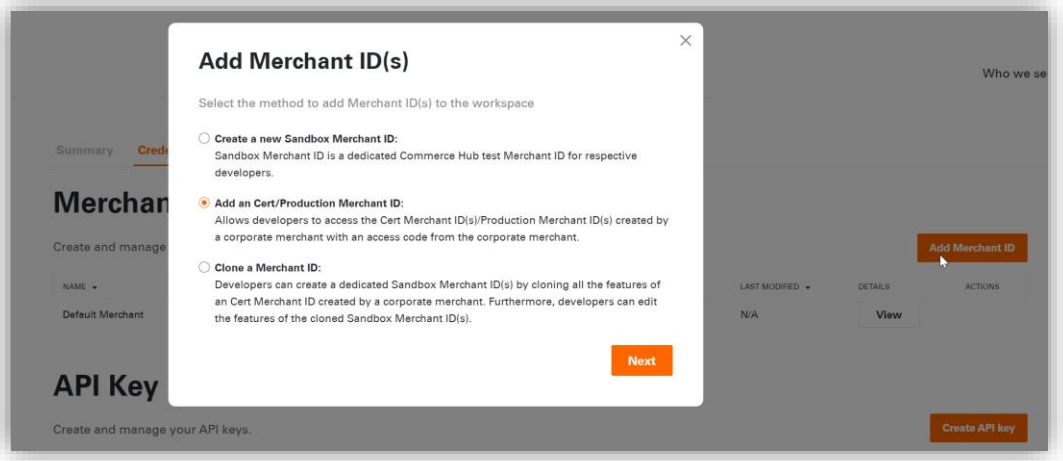
Click the **Credentials** tab:



This is where all of your API connections are created and managed. You will need a Merchant ID and an API Key.

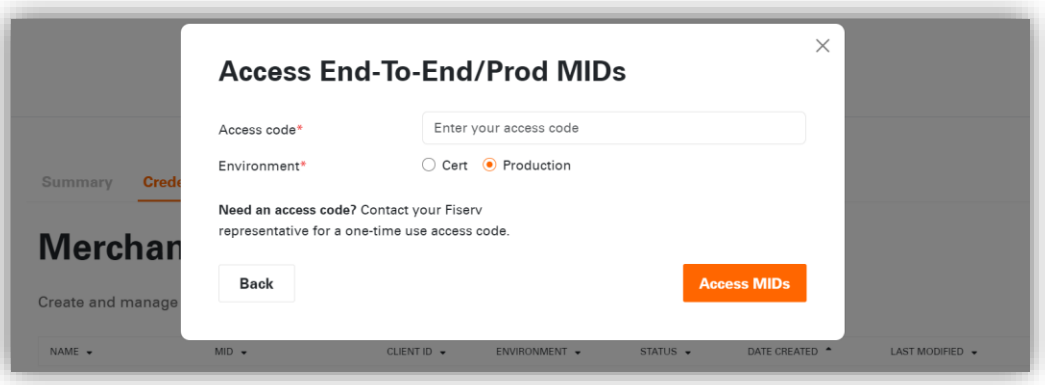


To connect your Carat account, click **Add Merchant ID**.



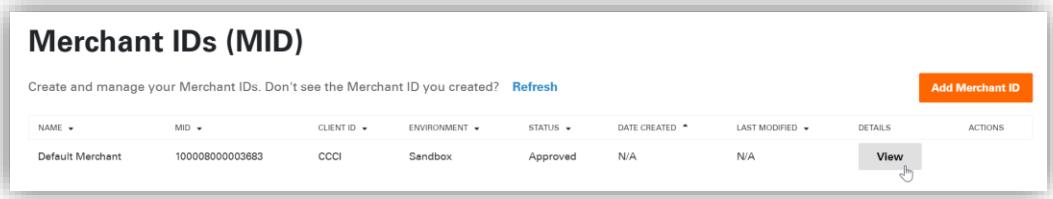
You will need an **Access code** to connect your Carat account. Please obtain this from your Fiserv representative.

If you are adding a Sandbox Merchant ID, you don't need an access code. The MID must be for **Card Not Present**, and must include the **CaptureType** and **Tokenization** features.

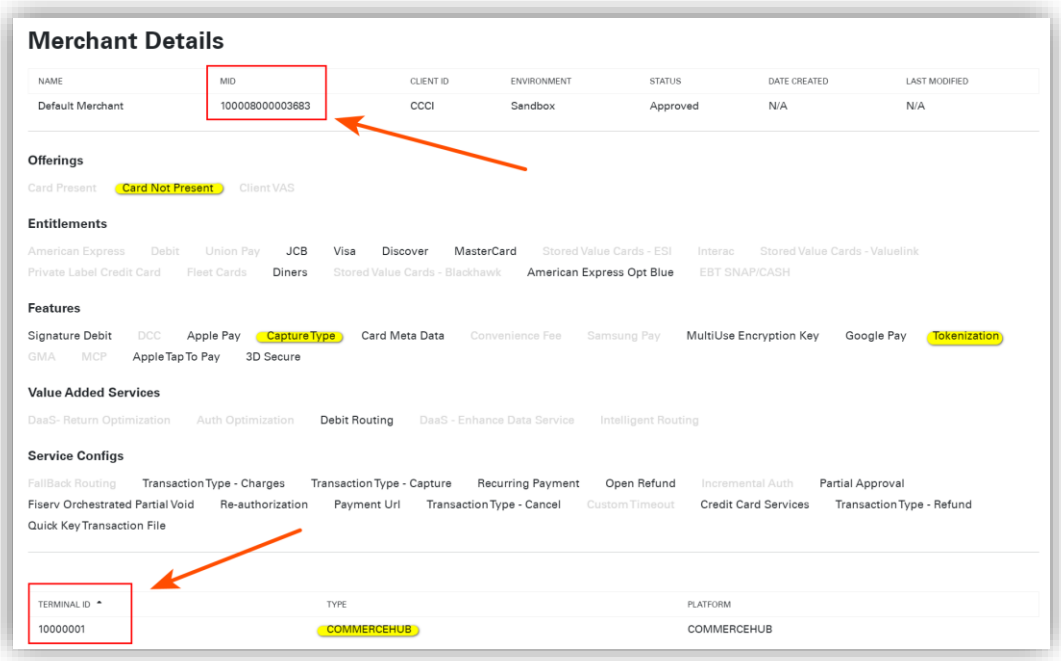


Once you've obtained and entered your Access code, continue through the process until you've successfully connected your Merchant ID.

View your Merchant ID:



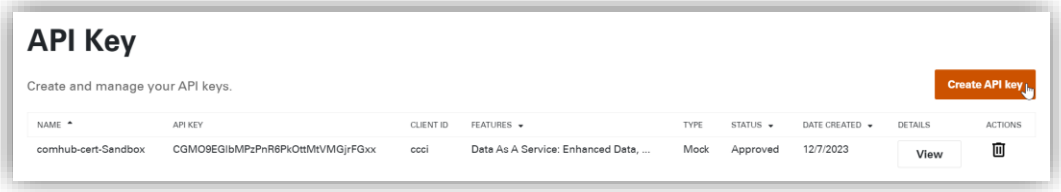
You need two values here. The first is your Merchant ID, listed as MID at the top. The second is your Terminal ID, at the bottom left.



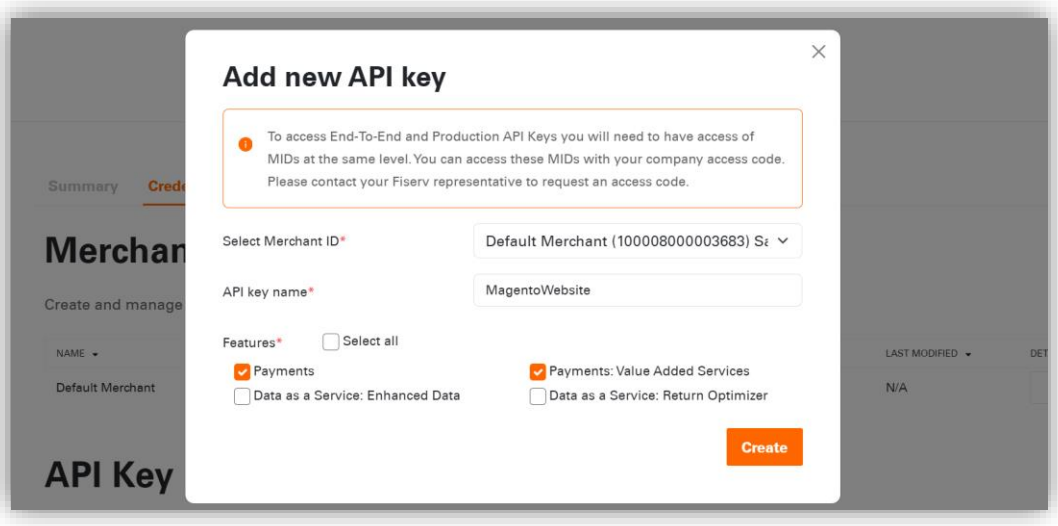
Copy the MID and Terminal ID into your Magento settings.

Now go back to the **Credentials** tab.

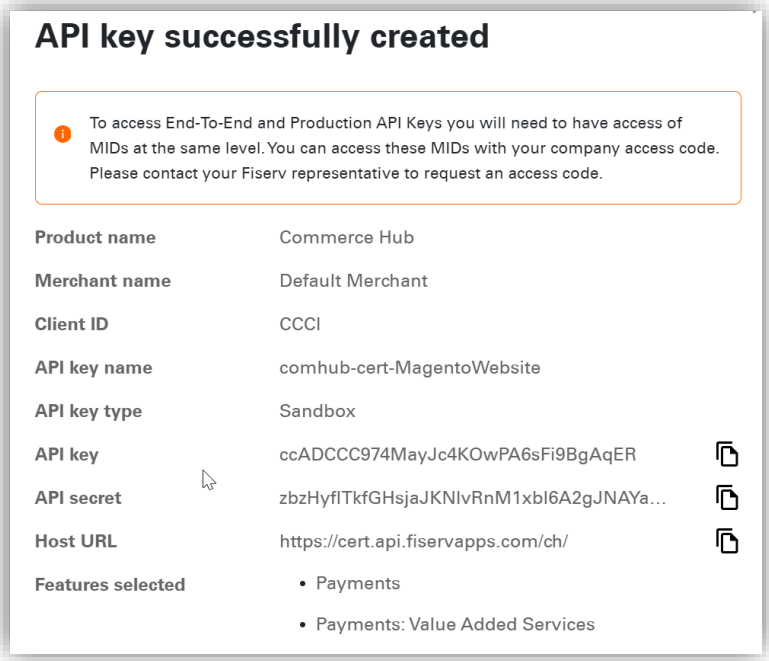
This time, click the **Create API Key** button:



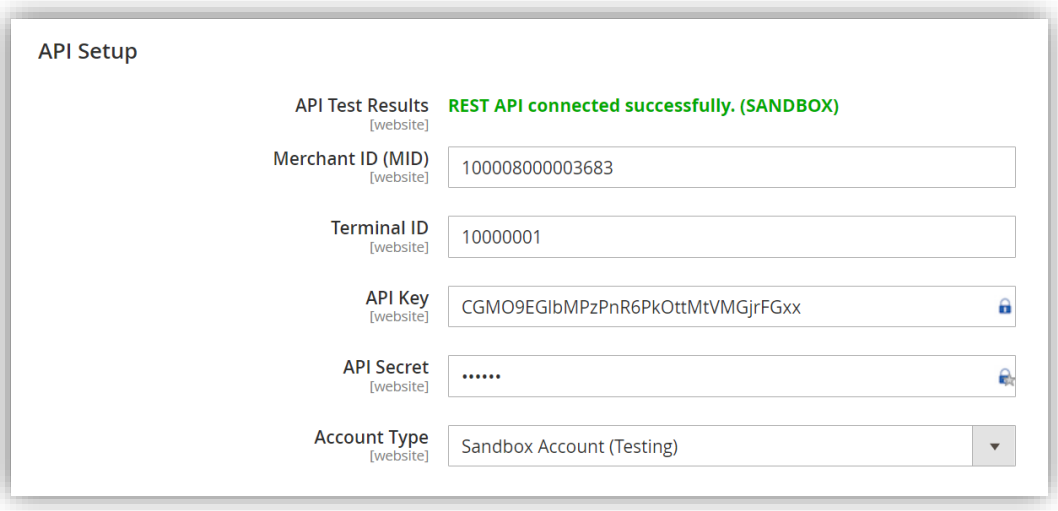
Complete the API Key form. Make sure the Features include **Payments**, and **Payments: Value Added Services**. You can include other features, but these two are the only ones currently necessary for this extension.



Once you click Create, you'll be given your new API Key. Copy the **API Key** and **API Secret** into your Magento Settings.



At this point, all of your API Setup fields should be completed. Set your **Account Type** to Sandbox or Production, based on the **API Key Type** you just created.



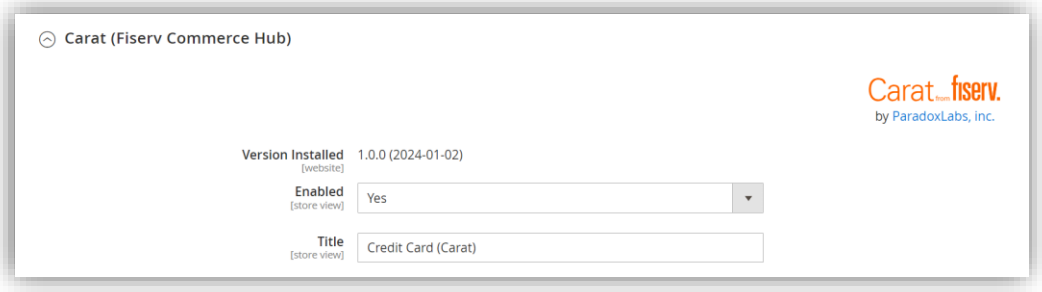
Once you've entered all of the values into Magento, click **Save Config**. Once the page reloads, you should have a green message: **REST API connected successfully.** If you get a red message, verify that you entered the values correctly and didn't miss any part of them.

Congratulations! You've completed connecting your new Magento extension to your Carat account. Please see the next section for information on the rest of the configuration options in Magento.

Configuration

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

General

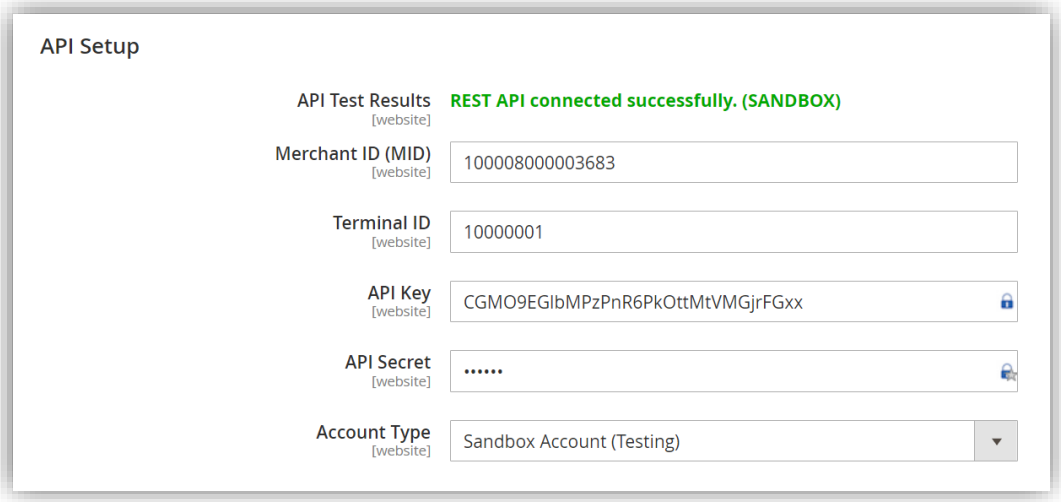


- **Version Installed**: This tells you the version of our extension currently installed on your website. Please include this in any support requests.
- **Enable**: Yes to enable the payment method. If disabled, you will still be able to invoice/refund existing orders, but it will not show up as a payment option during checkout.
- **Title**: This controls the payment option label on checkout and order status pages.

API Setup

This section connects your store to Carat for payment processing: Authorizations, captures, refunds, etc. You won't be able to process payments without it.

Please see Connecting A New Carat Account: Step 2. API Setup for details on finding and configuring these API credentials.

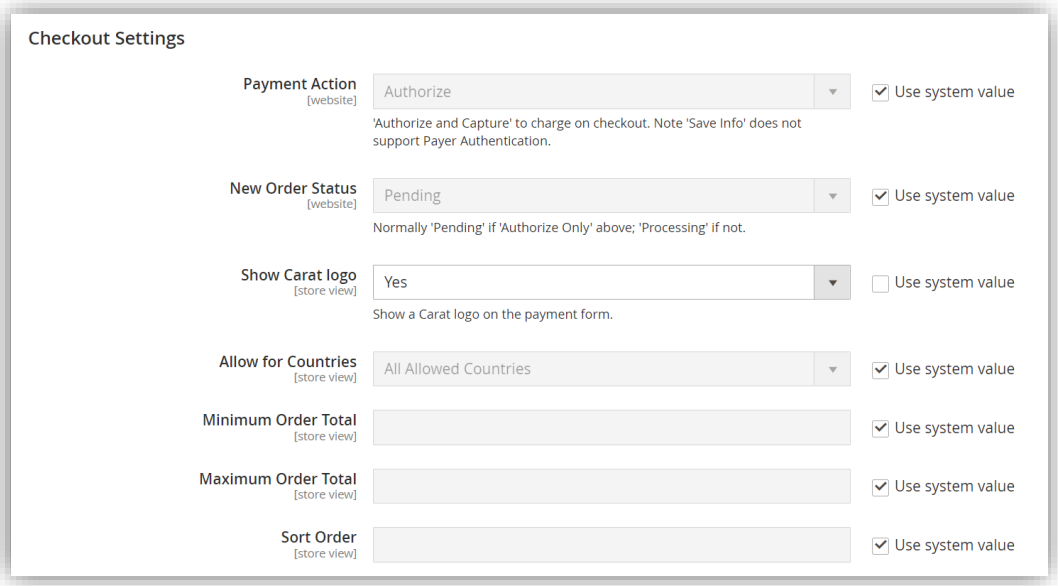


- **API Test Results**: Once you've completed and saved these settings, we will connect to Carat to verify that the connection works successfully. If we cannot connect to Carat, or your credentials are incorrect, this

will tell you with a red message. Correct the error, then reload the page and it should show **REST API connected successfully.**

- **Account Type**: If Sandbox, all requests will be made to Carat's test APIs, and no actual payments will be processed. If you want to test, you must have a sandbox Merchant ID and API Key. You can create them at: https://developer.fiserv.com

Checkout Settings

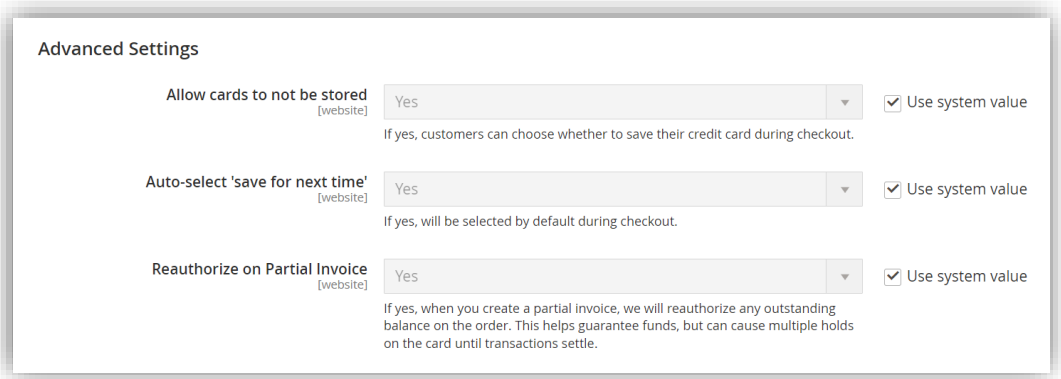


- **Payment Action**: Choose from the following options.
  - **Save info (do not authorize)**: This will require customers to enter a credit card on checkout, and store that credit card in Carat. The credit card will be validated in the process. No funds will be captured or held from the credit card upon checkout. Invoicing the order will perform a standalone authorize+capture transaction, but is not guaranteed to go through (funds may not be available).
  - **Authorize**: This will authorize the order amount upon checkout, allowing for manual invoicing and capture of the funds later. The authorized funds will be held (reserved) for about a week depending on your processor. If you do not invoice within that time, 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 Carat. Default Magento behavior is 'Pending' for Authorize Only, and 'Processing' for Authorize and Capture.
- **Show Carat logo**: If yes, checkout will display a 'Carat' logo above the payment form.
- **Allowed for Countries**: This setting allows you to limit which countries are allowed to use this payment method.
- **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.
- **Sort 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



- **Allow cards 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 Carat, 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 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).
- **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.
- **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. Any failure during reauthorization is ignored.

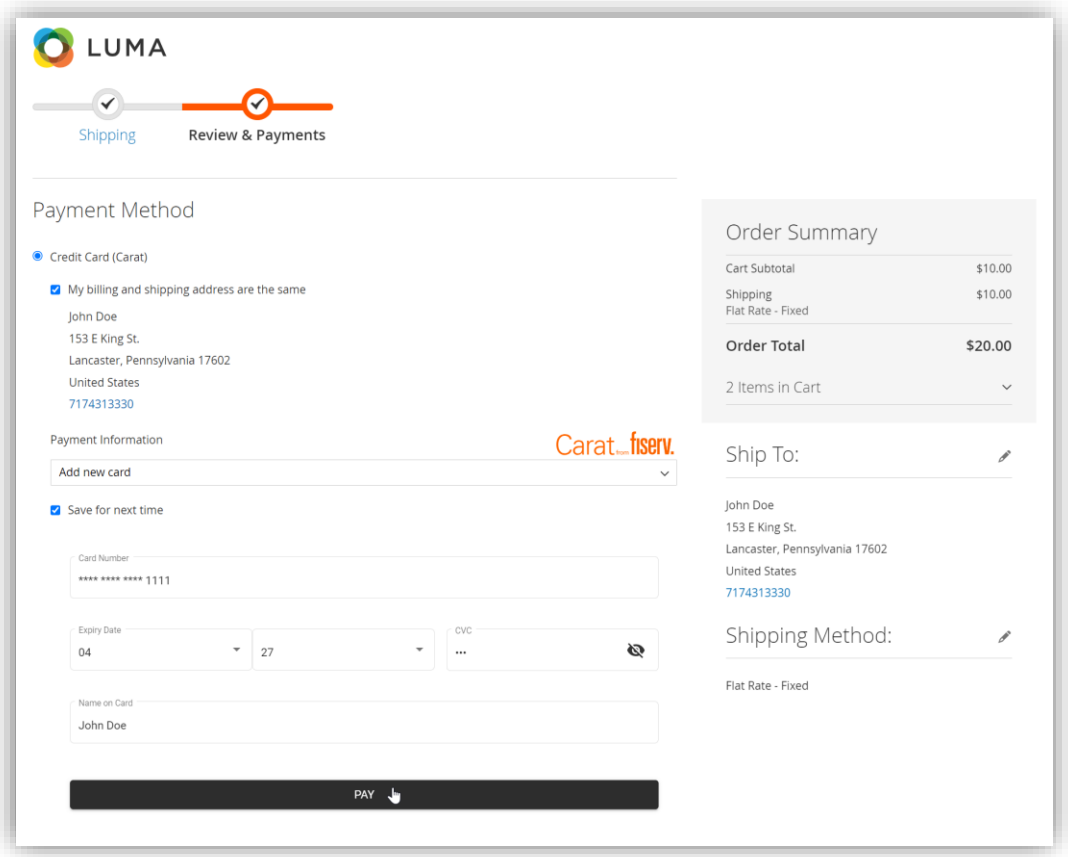
This concludes the payment method's configuration options.

Behavior Notes

For the most part this is a standard Magento payment method, with the addition of the advanced Stored Card functionality and everything related to that. However, there are some things worth note:

User Experience

All Carat credit card forms in the extension use an iframe hosted form. That greatly enhances security, but limits how much control you have over the payment form. For instance, the credit card form includes a 'Name on Card' field that is redundant to the billing address. There is no mechanism to prefill or disable that field.



The form has built-in security measures to prevent attack. If a customer reloads the payment form too quickly, they may find an error message instead: *"The transaction limit was exceeded. Please try again!"* – If this happens, the user should wait a minute and then try again (reload checkout).

Security

The Iframe Hosted Checkout form will always be active. There is no option to turn it off and revert to 'normal' credit card input. All credit card details (CC number, expiration date, CCV) are entered into an iframe hosted by Carat, and transmitted directly to Carat's servers. At no time will credit card numbers ever touch your server for this payment method. This allows for the best possible mix of user experience and PCI compliance.

Some data about credit cards is kept in your database. This includes the card type, last 4 digits, expiration date, and identifiers, among other data. All of this data is available via the Carat API, and no part of it is considered 'Confidential Cardholder Data' in the context of PCI compliance.

If you intend to collect payments from other sources using the Magento API, you will need to use a Carat API to tokenize the credit card, then store that card and token in Magento as a TokenBase Card, via the REST or GraphQL API. See the Technical / Integration Details section at the end for more information. This payment method is incapable of processing a transaction or storing a card using raw credit card details; it will not accept them under any circumstances.

Card Storage

Credit cards will always be stored in Carat to allow advanced functionality, even if the customer chooses not to save upon checkout. If the customer chooses not to save, or is a guest, the card will not show up for reuse within Magento. In this case, the card will be automatically purged from Magento after 120 days past its last use. This ensures the info is available for edits, captures, and refunds, but respects the customer' wishes.

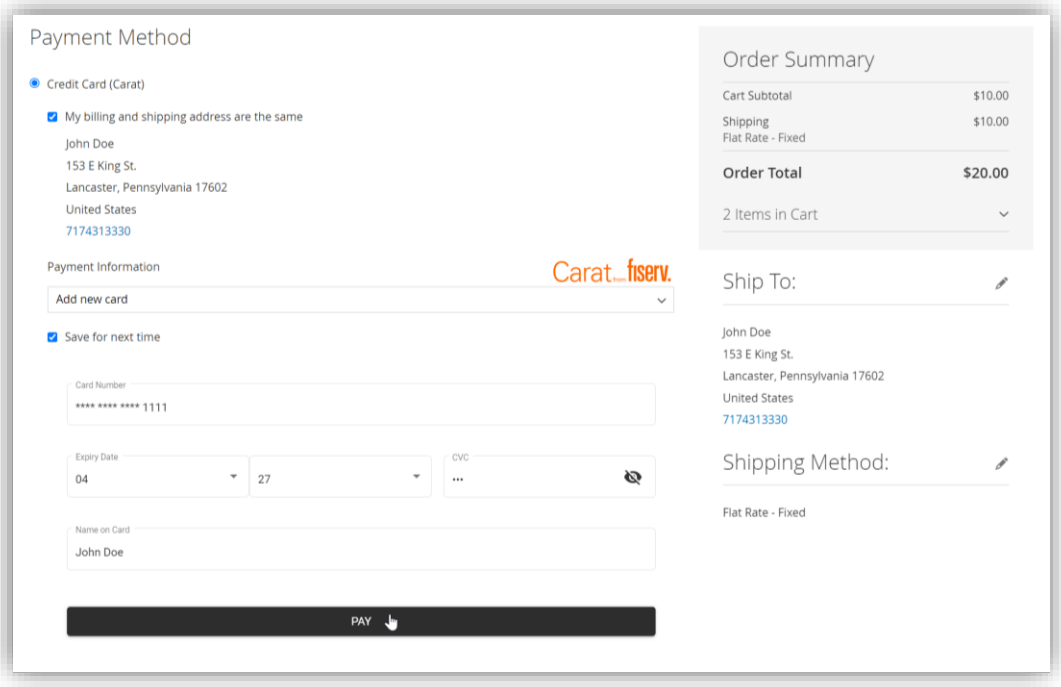
There is no automatic card duplicate prevention: If a customer enters the same credit card as a new card 3 times, it will be stored within Carat 3 times. There is no limit to the number of cards/tokens on a Carat account.

Usage

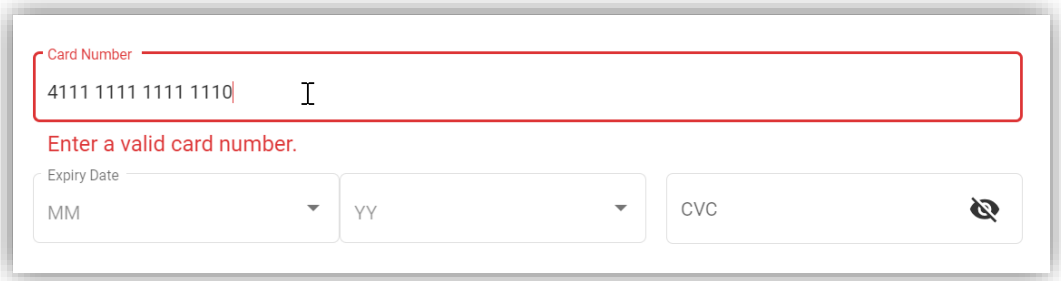
There isn't much to using this extension in practice: It's a standard Magento payment method, and all interfaces should be self-explanatory. Here's a rundown of what you get:

Checkout Payment Form

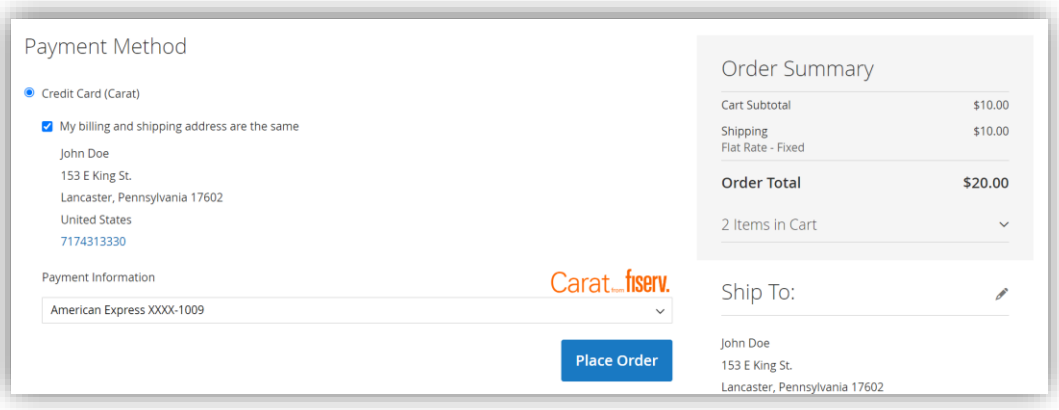
The frontend payment form lets you choose/enter billing address and credit card. You can choose an existing card (if any) from the dropdown, or to add a new one.



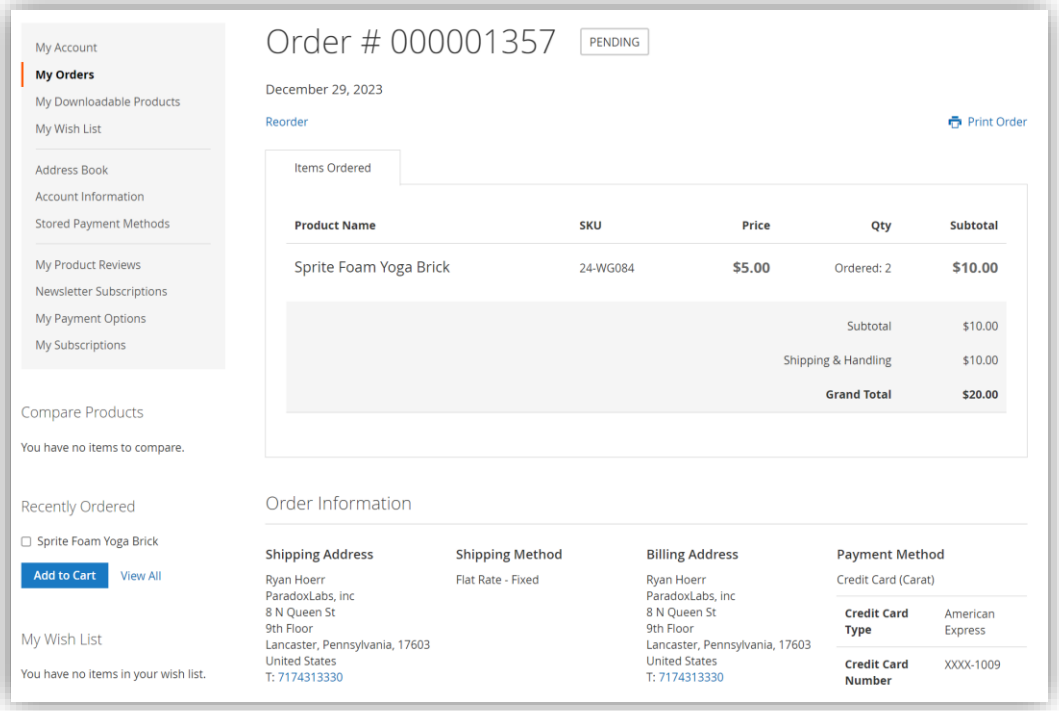
Credit card type is detected automatically, and any format errors are immediately displayed below the input field.



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

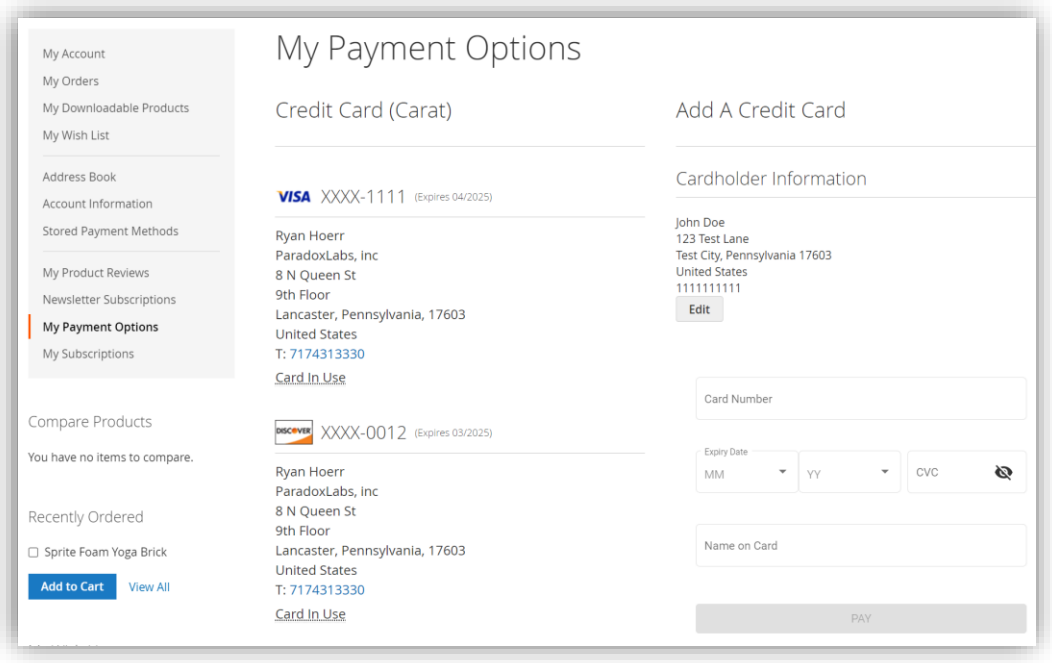


Order status page



Customer 'My Payment Options' account area

The My Payment Options section allows customers to see their stored cards, add, edit, and delete.



Like on checkout, the address must be entered and confirmed before payment info can be entered. This means all card adding/editing is a two step process. You can go back and edit the address after confirming it, but that will clear out any changes that may have been made to payment data.

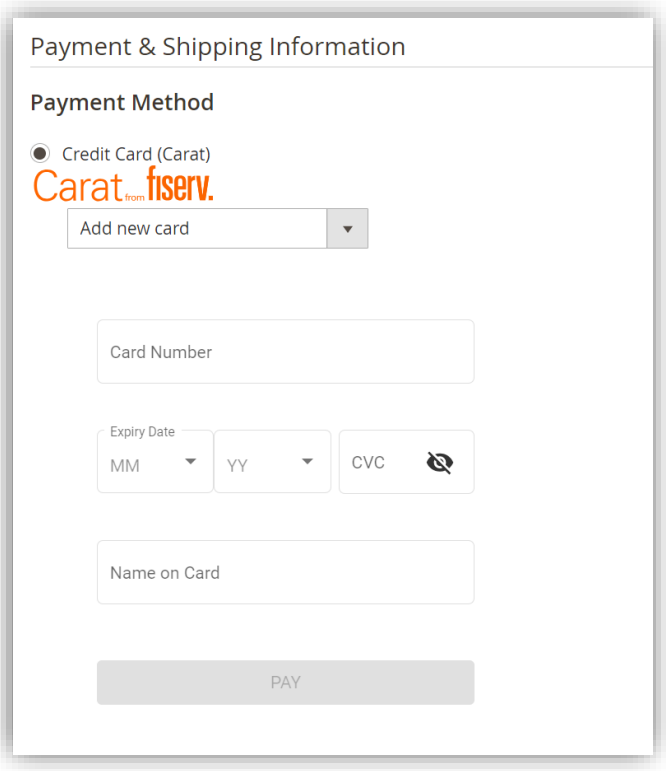
When editing a card, the full payment data will have to be reentered for any changes.

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, the Payment Data 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 if necessary; please contact us if you have a problem.

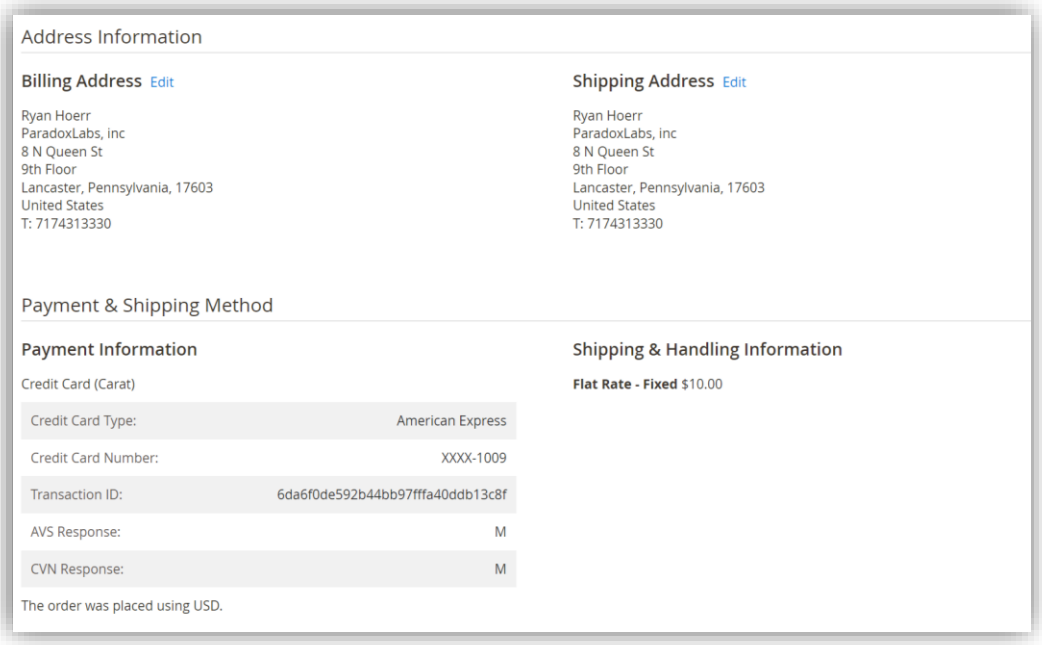
Admin order form

The admin form has the same options and behavior as frontend checkout.



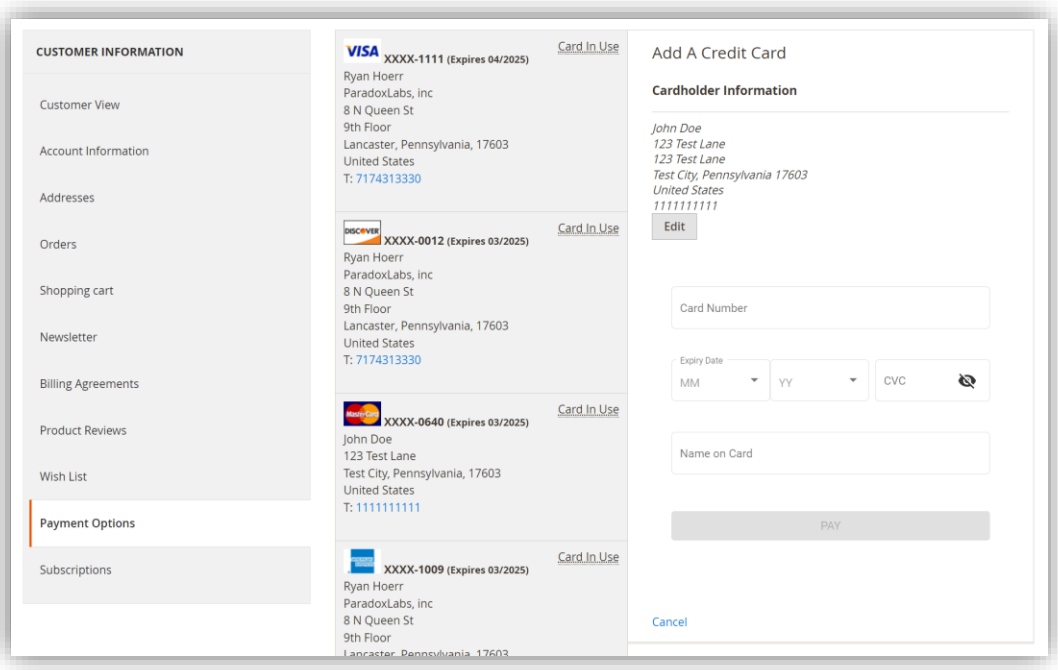
Admin order status page

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



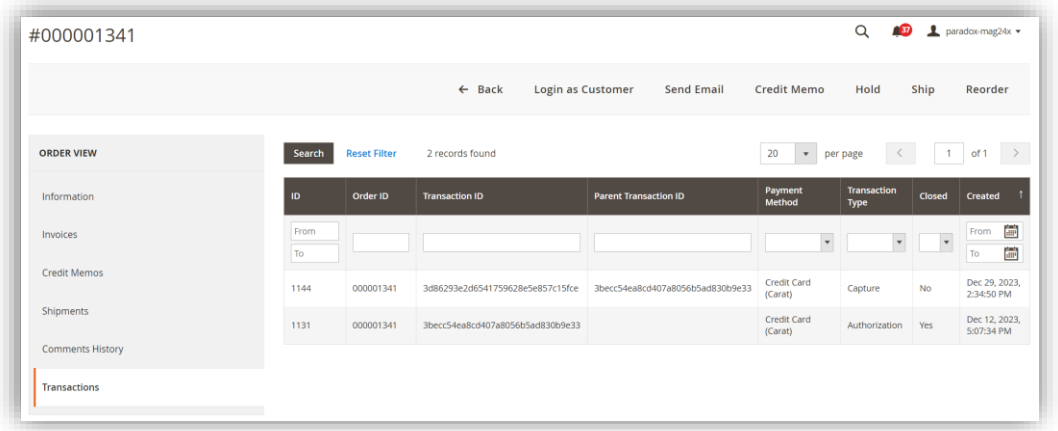
Admin customer ‘Payment Options’ account area

Viewing a customer, you will see an added ‘Payment Options’ tab. This shows all of the same information with all of the same functionality as the equivalent frontend section. You can use this to view, add, edit, and delete your customers’ stored Carat payment data.

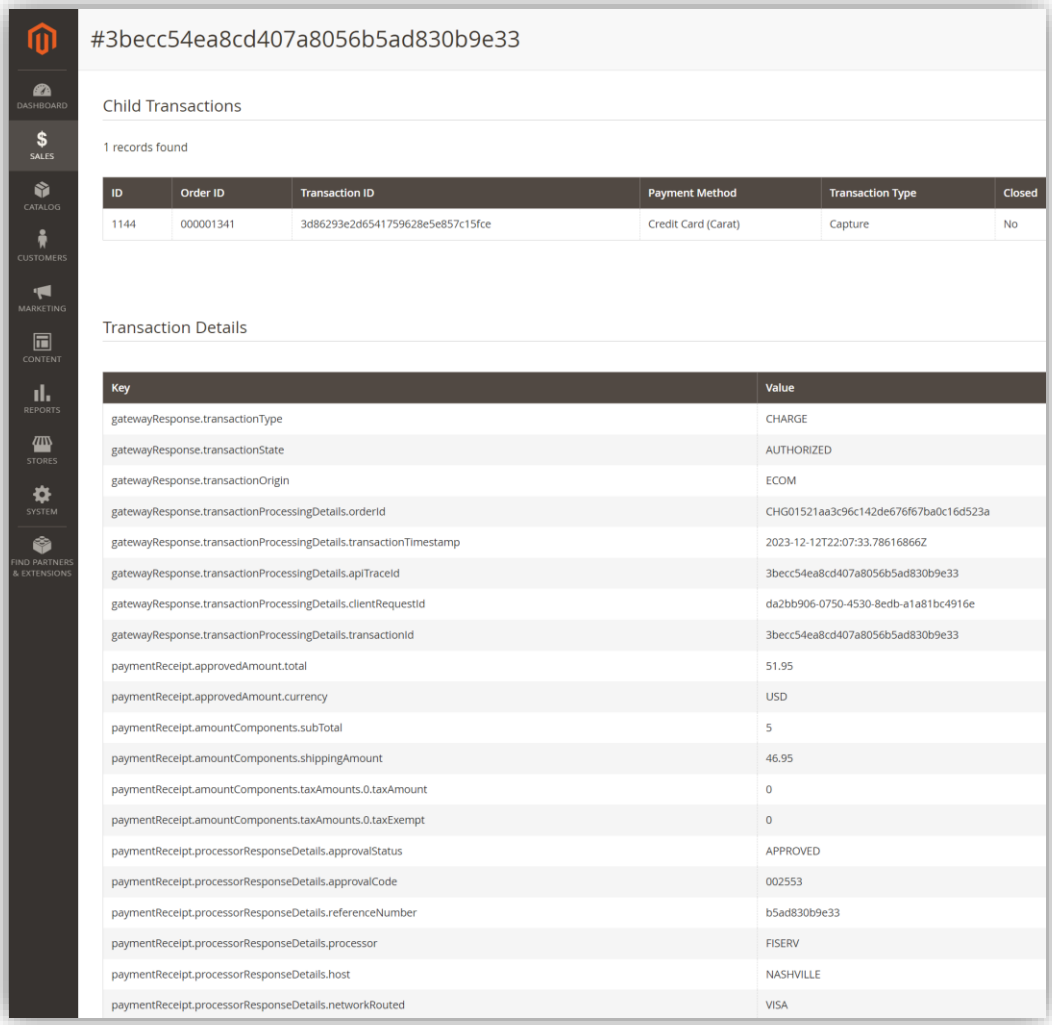


Admin transaction info

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 returned by the Carat Simple Order API. This same data is accessible from the transaction record in code.


#3becc54ea8cd407a8056b5ad830b9e33
DASHBOARD
SALES
CATALOG
CUSTOMERS
MARKETING
CONTENT
REPORTS
STORES
SYSTEM
FIND PARTNERS & EXTENSIONS
Child Transactions
1 records found
ID
Order ID
Transaction ID
Payment Method
Transaction Type
Closed
1144
000001341
3d86293e2d6541759628e5e857c15fce
Credit Card (Carat)
Capture
No
Transaction Details
Key
Value
gatewayResponse.transactionType
CHARGE
gatewayResponse.transactionState
AUTHORIZED
gatewayResponse.transactionOrigin
ECOM
gatewayResponse.transactionProcessingDetails.orderId
CHG01521aa3c96c142de676f67ba0c16d523a
gatewayResponse.transactionProcessingDetails.transactionTimestamp
2023-12-12T22:07:33.78616866Z
gatewayResponse.transactionProcessingDetails.apiTraceId
3becc54ea8cd407a8056b5ad830b9e33
gatewayResponse.transactionProcessingDetails.clientRequestId
da2bb906-0750-4530-8edb-a1a81bc4916e
gatewayResponse.transactionProcessingDetails.transactionId
3becc54ea8cd407a8056b5ad830b9e33
paymentReceipt.approvedAmount.total
51.95
paymentReceipt.approvedAmount.currency
USD
paymentReceipt.amountComponents.subTotal
5
paymentReceipt.amountComponents.shippingAmount
46.95
paymentReceipt.amountComponents.taxAmounts.0.taxAmount
0
paymentReceipt.amountComponents.taxAmounts.0.taxExempt
0
paymentReceipt.processorResponseDetails.approvalStatus
APPROVED
paymentReceipt.processorResponseDetails.approvalCode
002553
paymentReceipt.processorResponseDetails.referenceNumber
b5ad830b9e33
paymentReceipt.processorResponseDetails.processor
FISERV
paymentReceipt.processorResponseDetails.host
NASHVILLE
paymentReceipt.processorResponseDetails.networkRouted
VISA

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.

Is ParadoxLabs Carat Payments PCI Compliance?

PCI compliance is a complex and multifaceted issue, covering every aspect of your business. We can't guarantee that your business is PCI-compliant. That depends on your server, passwords, business processes, regular security scans, any other payment methods, and a lot more. What we can tell you is that this extension will not prevent you from being PCI compliant. We don't log confidential cardholder data or do anything else that would bring you under scrutiny.

This extension implements **Hosted Checkout** iframes for all credit card forms, and does not support collecting credit card data by any other means. That makes the ParadoxLabs Carat payment method eligible for **PCI v3.2 Self-Assessment Questionnaire A (SAQ A)**, the simplest possible SAQ form.

Note that you **must** have SSL (TLS) enabled on all checkout and login forms, and that this eligibility **only** applies to this specific payment method. Any other payment methods or credit card handling your business may perform will have its own SAQ eligibility, and may require you to complete a more stringent SAQ form (A-EP or D).

For details on the SAQ types and what eligibility means, see "Self-Assessment Questionnaire Instructions and Guidelines (3.2)" (PDF, by PCI Standards Security Council).

How do I do an online refund from Magento?

In order to process an 'online' refund through Carat, 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 capture transactions, but every capture has an invoice that's directly related. You refund an invoice, not an order.

How does this payment method handle currency?

Transactions are processed in the base currency for the website customers are purchasing from. Any alternate currencies selected on the frontend are converted to the website's base currency by Magento's built-in currency handling, based on conversion rates provided by a web service configured in your Magento Admin Panel.

Magento allows for a separate base currency per website, if configured to do so. In order to define explicit prices in multiple currencies, each currency must have its own website where it is set as the base currency. All currency setup is configured outside of our payment extension settings.

Your Carat account must allow transactions to be processed in your website's base currency(s).

Technical / Integration Details

Architecture

The payment method code for this method is `paradoxlabs_carat`.

`ParadoxLabs_Carat` 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\Carat\Model\Method`. This talks to Carat primarily through `\ParadoxLabs\Carat\Model\Gateway` (via Commerce Hub REST API), and stores card metadata in instances of `\ParadoxLabs\Carat\Model\Card`. Each of these extends an equivalent abstract class in TokenBase, and implements only the details specific to the Carat API.

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

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

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 a new TokenBase Card with a valid Carat credit card token (and associated card and address data). 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. You must store the card in Carat via a form session, then tokenize that session, then store the resulting token as a card, prior to using it for an order. See Carat documentation on how to tokenize a credit card in various circumstances.

The extension's custom REST API requests are detailed 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": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2022",
        "cc_exp_month": "12",
        "cc_bin": "400700"
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "Visa XXXX-0027"
}
```

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": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
        "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": "01",
        "cc_exp_year": "2023",
        "cc_last4": "0027",
        "cc_type": "VI",
        "cc_bin": "400700"
    }
}
```

Example response:

```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_country": "US",
        "cc_bin": "400700"
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": "paradoxlabs_carat",
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
        "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": "01",
        "cc_exp_year": "2023",
        "cc_last4": “0027”,
        "cc_type": “VI”,
        "cc_bin": "400700"
    }
}
```


Example response:


```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": “01”,
        "cc_bin": "400700"
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_bin": "400700"
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
        "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_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    }
}
```

Example response:

```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
```

```
        "hash": "50b8e326b012e793957215c0361afc4b52434b26",
        "active": "1",
        "expires": "2023-01-31 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_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    }
}
```

Example response:

```
{
    "id": 1,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": "VI",
        "cc_last4": "0027",
        "cc_exp_year": "2023",
        "cc_exp_month": "01",
        "cc_bin": "400700"
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": "paradoxlabs_carat",
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```

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": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
        "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_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    }
}
```

Example response:

```
{
    "id": 95,
    "in_use": false,
    "additional_object": {
        "cc_type": “VI”,
        "cc_last4": “0027”,
        "cc_exp_year": “2023”,
        "cc_exp_month": "01",
        “cc_bin”: “400700”
    },
    "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": null,
    "payment_id": "8405263549590027",
    "method": “paradoxlabs_carat”,
    "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": "2023-01-31 23:59:59",
    "label": "Visa XXXX-0027"
}
```


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 the Chrome Altair GraphQL Client 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 `paradoxlabs_carat`; 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.

CaratSecureParams(input: TokenBaseCaratSecureInput!): TokenBaseCaratSecureParams
Get the Carat Secure hosted form parameters for collecting payment data. Takes a cart ID (if on checkout). Response data is the parameters needed to initialize the Fiserv Carat hosted form client JS SDK.

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
        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
}
```

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
}
```

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 {
        save: Boolean                                   Save the card for later use?
        card_id: String                                 Card identifier hash to use
        session_id: String                              Carat secure form session ID
}
```

TokenBaseCaratSecureInput

Input for fetching a Secure form parameters. The form location affects whether the form is tied to the cart session data (billing address, etc.) or not.

```
input TokenBaseCaratSecureInput {
        cardId: String                                  Card ID hash
}
```

TokenBaseCaratSecureParams

Data necessary to load a Secure payment form. This consists of JSON-encoded parameters. The parameters must be decoded and then passed to the Carat client JS SDK in order to load the payment form.

```
type TokenBaseCaratSecureParams {
        iframeParams: String                            Parameters for the Secure form (JSON)
}
```

GraphQL Query Examples

Some response data has been omitted for brevity.

Fetch card by ID

Example request:

```
{
  tokenBaseCards(hash:"ec431a3e1f9904a35dc083a257cf2585de7b7b6c") {
    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_bin,
      cc_last4,
      cc_exp_year,
      cc_exp_month
    }
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCards": [
      {
        "label": "Visa XXXX-0027",
        "expires": "2022-12-31 23:59:59",
        "hash": "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
        "customer_email": "roni_cost@example.com",
        "customer_id": 1,
        "profile_id": null,
        "payment_id": "8405263549590027",
        "method": “paradoxlabs_carat”,
        "active": true,
        "created_at": "2020-02-25 18:05:12",
        "updated_at": "2020-02-25 18:05:12",
        "last_use": null,
        "address": {
          "region": {
            "region_code": "PA",
            "region": "Pennsylvania",
            "region_id": 51
          },
          "region_id": 51,
          "country_id": "US",
          "street": [
            "123 Test Lane",
            ""
          ],
          "company": "",
          "telephone": "1111111111",
          "postcode": "17603",
          "city": "Test City",
          "firstname": "John",
          "lastname": "Doe"
        },
        "additional": {
          "cc_type": "VI",
          "cc_bin": "400700",
          "cc_last4": "0027",
          "cc_exp_year": "2022",
          "cc_exp_month": "12"
        }
      }
    ]
```

```
  }
}
```

Fetch checkout config

Example request:

```
{
  tokenBaseCheckoutConfig(method:"paradoxlabs_carat") {
    method,
    useVault,
    canSaveCard,
    forceSaveCard,
    defaultSaveCard,
    logoImage,
    paramUrl
  }
}
```

Example response:

```
{
  "data": {
    "tokenBaseCheckoutConfig": {
      "method": "paradoxlabs_carat",
      "useVault": true,
      "canSaveCard": true,
      "forceSaveCard": false,
      "defaultSaveCard": true,
      "logoImage": "false",
      "paramUrl": "https://example.com/pdl_carat/secure/getParams/"
    }
  }
}
```

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: "paradoxlabs_carat",
      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: {
        session_id: "7f58b566-911d-4307-9e51-758391728c9a",
        save: true

      }
    }
  ) {
    label,
    expires,
    hash,
    customer_email,
```

```
    customer_id,
    customer_ip,
    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_bin,
      cc_last4,
      cc_exp_year,
      cc_exp_month
    }
  }
}
```

Example response:

```
{
  "data": {
    "createTokenBaseCard": {
      "label": "Visa XXXX-0027",
      "expires": "2022-12-31 23:59:59",
      "hash": "a5412c321a5de0c1f3964492e0bfba2b48e5984b",
      "customer_email": "roni_cost@example.com",
      "customer_id": 1,
      "customer_ip": "127.0.0.1",
      "payment_id": "8405263549590027",
      "method": “paradoxlabs_carat”,
      "active": true,
      "created_at": null,
      "updated_at": null,
      "last_use": null,
      "address": {
        "region": {
          "region_code": "PA",
          "region": "Pennsylvania",
          "region_id": 51
        },
        "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_bin": "400700",
        "cc_last4": "0027",
        "cc_exp_year": "2022",
        "cc_exp_month": "12"
      }
    }
```

```
  }
}
```

Delete card

Example request:

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

Example response:

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

Get Secure form params

Example request:

```
query {
  CaratSecureParams(input: {
    cartId: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
  }) {
    iframeParams
  }
}
```

Example response:

```
{
  "data": {
    "CaratSecureParams": {
      "iframeParams": "{...}"
    }
  }
}
```

Place an order

Example request:

```
mutation {
  setPaymentMethodOnCart(
    input: {
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
      payment_method: {
        code: "paradoxlabs_carat",
        tokenbase_data: {
          card_id: "ec431a3e1f9904a35dc083a257cf2585de7b7b6c"
        }
      }
    }
  ) {
    cart {
      selected_payment_method {
        code,
        tokenbase_card_id,
        tokenbase_data {
          cc_type,
          cc_last4,
          cc_exp_year,
          cc_exp_month
        }
      }
    }
  }
  placeOrder(
    input: {
```

```
      cart_id: "kDyOEKkJmIOxa6H3ceus6MjMaSF9lao1"
    }
  ) {
    order {
      order_number
    }
  }
}
```

Example response:

```
{
  "data": {
    "setPaymentMethodOnCart": {
      "cart": {
        "selected_payment_method": {
          "code": “paradoxlabs_carat”,
          "tokenbase_card_id": "ec431a3e1f9904a35dc083a257cf2585de7b7b6c",
          "tokenbase_data": {
            "cc_type": "VI",
            "cc_last4": "0027",
            "cc_exp_year": "2022",
            "cc_exp_month": "12"
          }
        }
      }
    },
    "placeOrder": {
      "order": {
        "order_number": "000000676"
      }
    }
  }
}
```

How-To: API Checkout Flow

The checkout flow for this extension is a little complex, due to the security mechanisms involved.

Step 1: Fetch checkout payment parameters

For GraphQL, see the ‘*Fetch checkout config*’ example. For standard checkout’s REST requests, these parameters are provided to JS by Magento’s frontend layout system.

There are several parameters that are critical to the Carat checkout flow. Most important is the Secure params URL:

```
      "paramUrl": "https://example.com/pdl_carat/secure/getParams/"
```

The other parameters communicate Magento payment settings and state, and can be used or disregarded as you choose.

Step 2: Fetch Secure Payload

When the time comes to initiate the credit card form, you will need to query `caratSecureParams`. We do this any time the ‘new card’ option is selected on our payment method, or payment method visibility changes, or the Secure form is completed or reset.

For GraphQL, see the *‘Get Secure Acceptance form’* example. The query takes many of the same inputs, but the keys differ. The response has iframeParams JSON-encoded. See the example and GraphQL schema for specifics.

If successful, this request will return a JSON payload like:

```
{
    "iframeParams": {
        "formconfig": {
            “merchantId”: “1234567890”,
            “publicKey”: “...”,
            “asymmetricEncryptionAlgorithm”: “RSA”,
            “keyId”: “...”
        },
        “authorization”: “...”,
        “apiKey”: “...”,
        “sessionId”: “...”,
    }
}
```

If an error occurs, it will return a 400 response code, and may provide a JSON response including error message.

Step 3: Initialize Secure Form

Once you have the Secure payload (`iframeParams`), you need to initialize the secure form itself. This requires loading the Fiserv Carat client JS SDK, and then instantiating it with these params. For example:

```
const form = new this.commercehub.Fiserv(
    data.formConfig,
    data.authorization,
    data.apiKey
);

form.loadPaymentForm(this.options.target)
    .then(this.handleSuccess.bind(this))
    .catch(this.handleError.bind(this));
```

Step 4: Receive form success

If successful, your handleSuccess method will be called when the form is completed (payment details entered) successfully. Complete the order by passing the sessionId as the order payment data.

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