Adaptive Subscriptions: User Manual

Version 3.7 – For Magento® 2.3+ – Updated 2024-06-28

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/subscriptions:*
php bin/magento module:enable -c ParadoxLabs_TokenBase ParadoxLabs_Subscriptions
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_Subscriptions
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 Adaptive Subscriptions

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
```

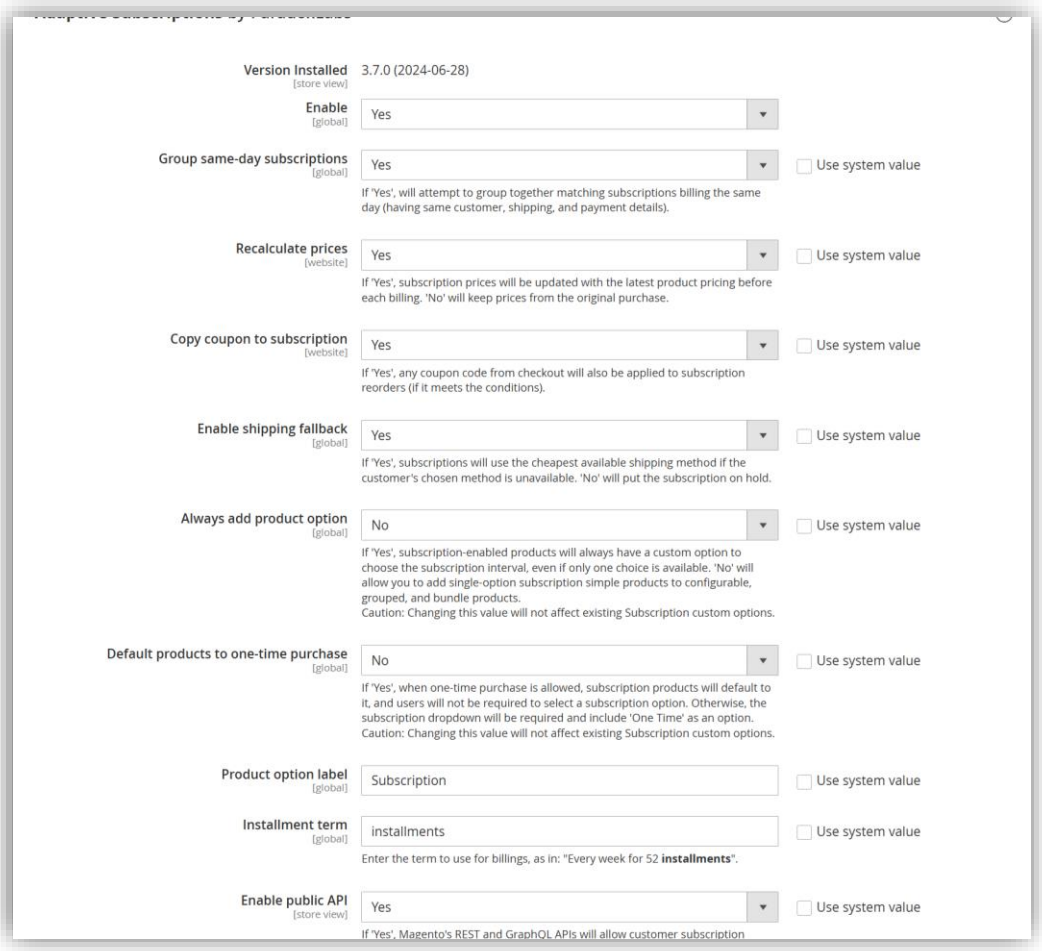
If your site is in production mode, you will also need to run these commands to recompile sources:

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

Configuration

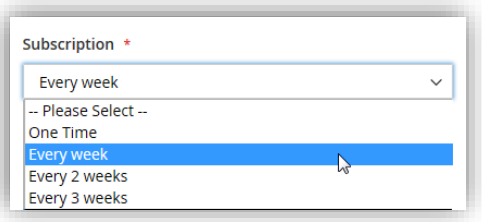
Open your Admin Panel and go to **Admin > Stores > Settings > Configuration > Catalog > Adaptive Subscriptions**. You'll find a settings page like the below.

General

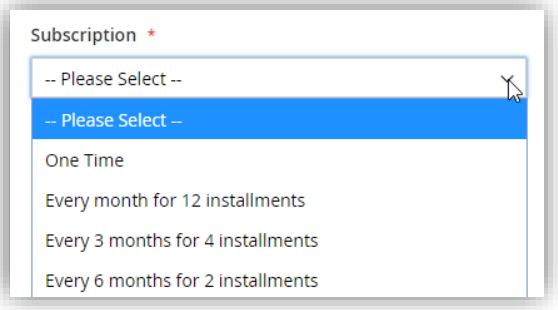


- **Enable**: Yes to enable the subscriptions module. If disabled, the tabs and grids will disappear, and subscription generation will stop.
- **Group same-day subscriptions**: If yes, scheduled billing will look for any matching subscriptions due the same day, and run them together in the same order. A matching subscription is one that is due the same day, with identical customer, billing, payment, and shipping information.
- **Recalculate prices**: If 'Yes', subscription prices will be updated with the latest product pricing each time a subscription renews. 'No' will keep subscription pricing from the original purchase (grandfathering the pricing indefinitely). If enabled, as a merchant, it's your responsibility to inform customers of price changes that will impact them.

- **Copy coupon to subscription**: If 'Yes', any coupon code from checkout will also be applied to subscription renewals (if it meets the sales rule conditions). If 'No', coupon codes will only ever apply at checkout. Changing this setting will only apply to new subscriptions placed thereafter; it is not retroactive.
- **Enable shipping fallback**: If yes, when the subscription's original/chosen shipping method is unavailable, it will automatically use the cheapest available shipping method instead. If no, the subscription will be put on hold if its assigned shipping method is not available at time of billing.
- **Always add product option**: If yes, subscription products will always have a custom option added to choose the subscription interval, even if only one choice is available. If you set this to 'no', you can add single-option subscription simple products to configurable, grouped, and bundle products. In that situation, communicating the subscription details to the customer is up to you.
- **Default products to one-time purchase**: If yes, when one-time purchasing is allowed for a product, those products will default to it and users will not be required to select an option to add it to cart. Otherwise, the subscription custom option dropdown will be required, and users can select 'One-Time' to purchase it without any renewal.
- **Product option label**: This text ("Subscription") is set as the default custom option label for each subscription product:



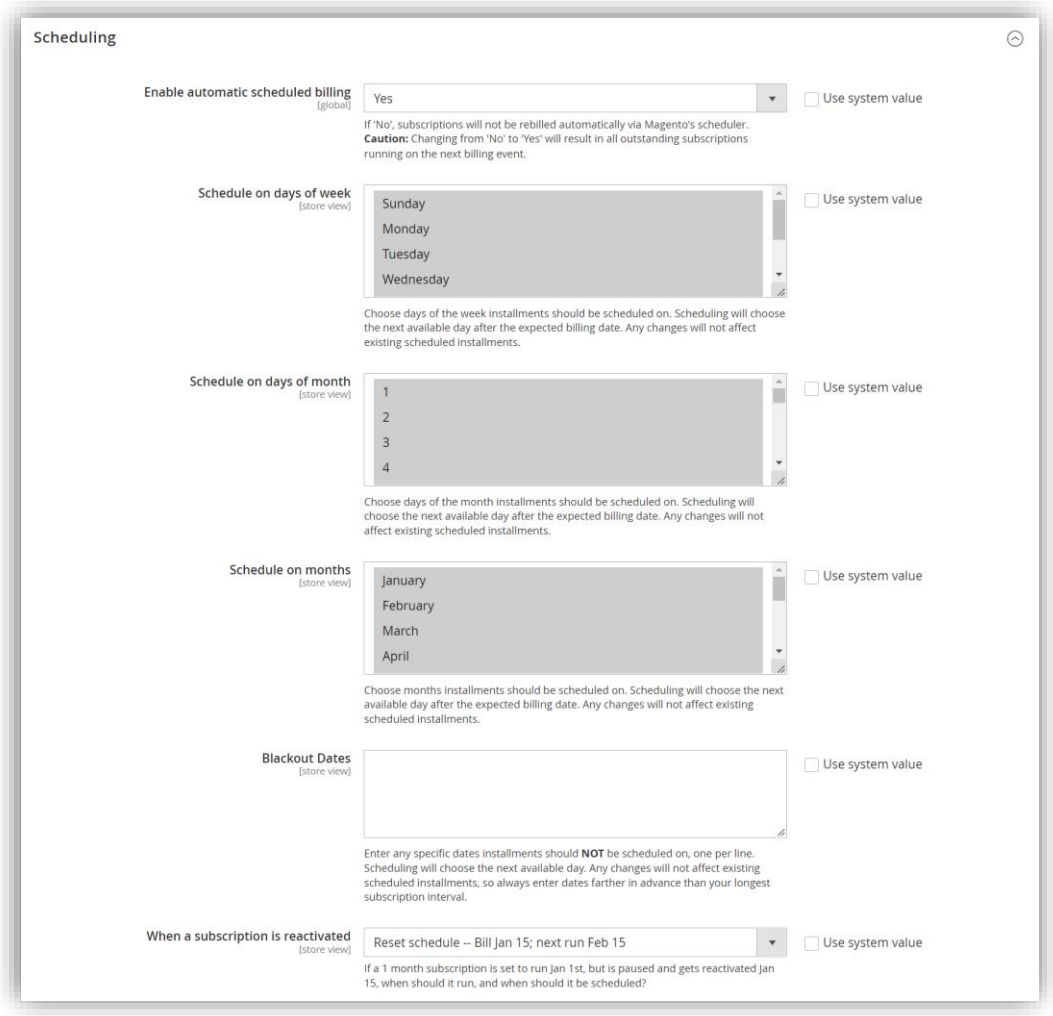
- **Installment term**: This text ("installments") is used to refer to subscription recurrences in several locations, particularly length-limited subscription options:



- **Enable public API**: If yes, Magento's REST and GraphQL APIs will allow customer subscription management. We recommend leaving this disabled unless you use them (most will not).

Scheduling

This section allows you to customize how subscription scheduling and billing occurs.



- **Enable automatic scheduled billing**: Yes to enable automatic billing via Magento's cron scheduler. If disabled, subscriptions will never run unless you trigger them manually.
- **Schedule on days of week**: Choose days of the week installments should be scheduled on. For example, if you deselect Saturday and Sunday, subscriptions will only schedule on Monday through Friday.
- **Schedule on days of month**: Choose days of the month installments should be scheduled on.
- **Schedule on months**: Choose months installments should be scheduled on.
- **Blackout dates**: Enter any specific dates installments should **NOT** be scheduled on, one per line. Use this to skip holidays, etc. Note: Any changes will not affect existing scheduled installments, so always enter blackout dates farther in advance than your longest subscription interval.
- **When a subscription is reactivated**: Choose from the following options:
    - **Keep existing**: This will leave the next run date unchanged (default). If a monthly subscription is set to run Jan 1st, but is paused and then reactivated on Jan 15th, it will rebill Jan 15th, and its next run date will be Feb 1st (Jan 1st + 1 month).
    - **Reset schedule**: This will reset the next run date to the current date, and shift future rebillings accordingly. If a monthly subscription is set to run Jan 1st, but is paused and then reactivated on Jan 15th, it will rebill Jan 15th, and its next run date will be Feb 15th (Jan 15th + 1 month).

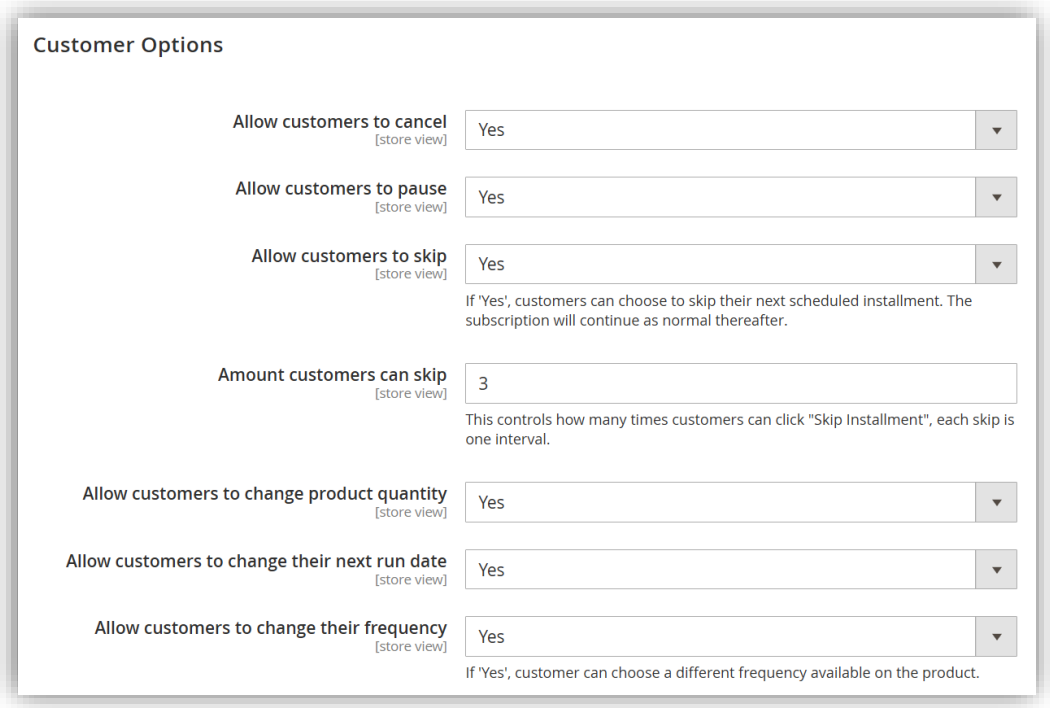
- **Recalculate**: This will recalculate the next run date for the next future date on the current schedule. If a monthly subscription is set to run Jan 1st, but is paused and then reactivated on Jan 15th, its next run date will be changed to Feb 1st (Jan 1st + 1 month). It will not bill until Feb 1st.

See the

Billing and Scheduling section for more info on how scheduling occurs.

Customer Options

This section is for toggling customer abilities.

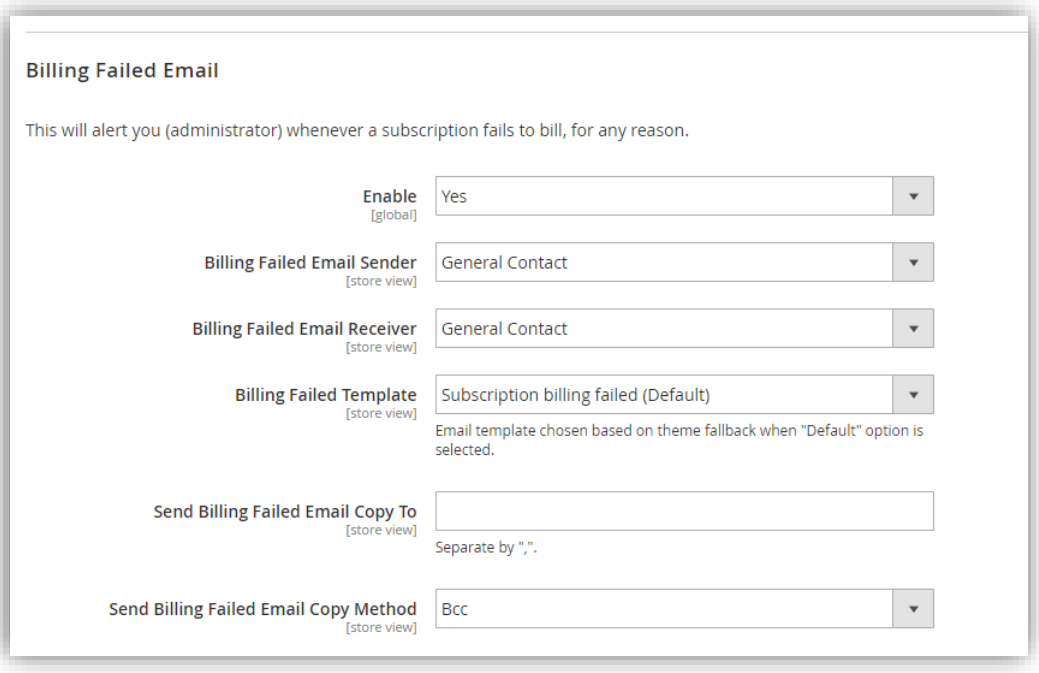


- **Allow customers to cancel**: If yes, customers will have a 'cancel' button when viewing their subscription (if it is not already canceled or complete). Once canceled, you cannot rebill, edit, or reactivate a subscription. If no, the customer will have to contact you to cancel.
- **Allow customers to pause**: If yes, customers will have a 'pause' button when viewing their subscription (if it is active). While paused, the subscription will not rebill. The customer will be able to reactivate it themselves. If no, the customer will have to contact you to pause.
- **Allow customers to skip**: If yes, customers will have a 'skip' button when viewing their subscription (if allowed). Clicking 'skip' will move the next run date back by one interval. This can be done a certain number of times at once, up to the limit set in the next setting.
- **Amount customers can skip**: This controls how many times a customer can delay their subscription's next run date at once. For example, if this is set to 3 for a monthly subscription, in June the customer can delay it up to 3 times for a next run date in September, but no further. If you set it to 1, the customer will only be able to delay one installment at a time from the current date.
- **Allow customers to change product quantity**: If yes, the customer can change the quantity of the product in their subscription from the frontend subscription edit form. This will update the subscription's pricing.
- **Allow customers to change their next run date**: If yes, the customer can change their subscription's next run date to any future day. Their selection will still be subject to your scheduling settings, so may not run on the exact date chosen.
- **Allow customers to change their frequency**: If yes, the customer can change their subscription frequency to any other option available for the product. This will update the subscription's pricing. It will not change

the subscription's billing count or length. If you use limited-duration subscriptions, you should keep this disabled to avoid unexpected behavior.

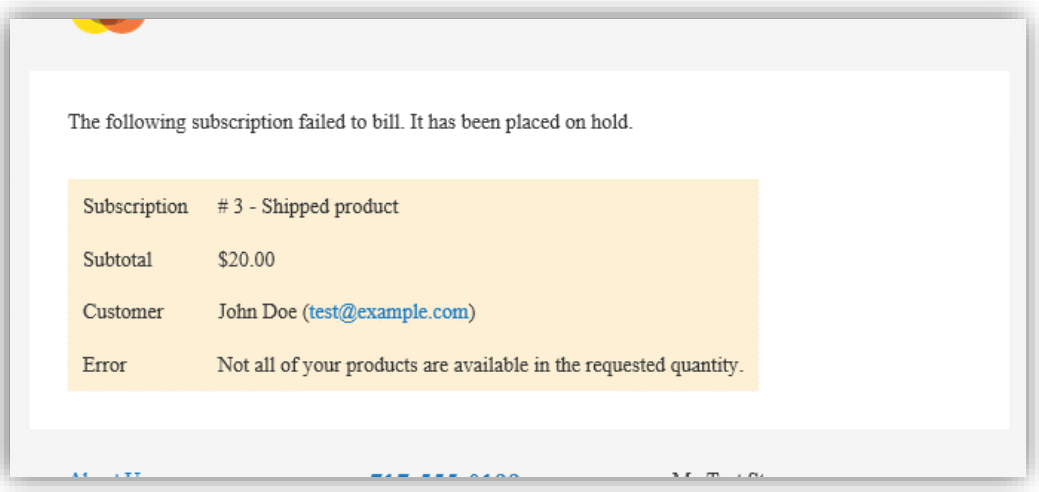
Billing Failed Email

This section allows you to control the billing failure notification. This is sent to an administrator any time a subscription fails to bill, for any reason: Payment failure, product disabled or out of stock, shipping method unavailable, etc.



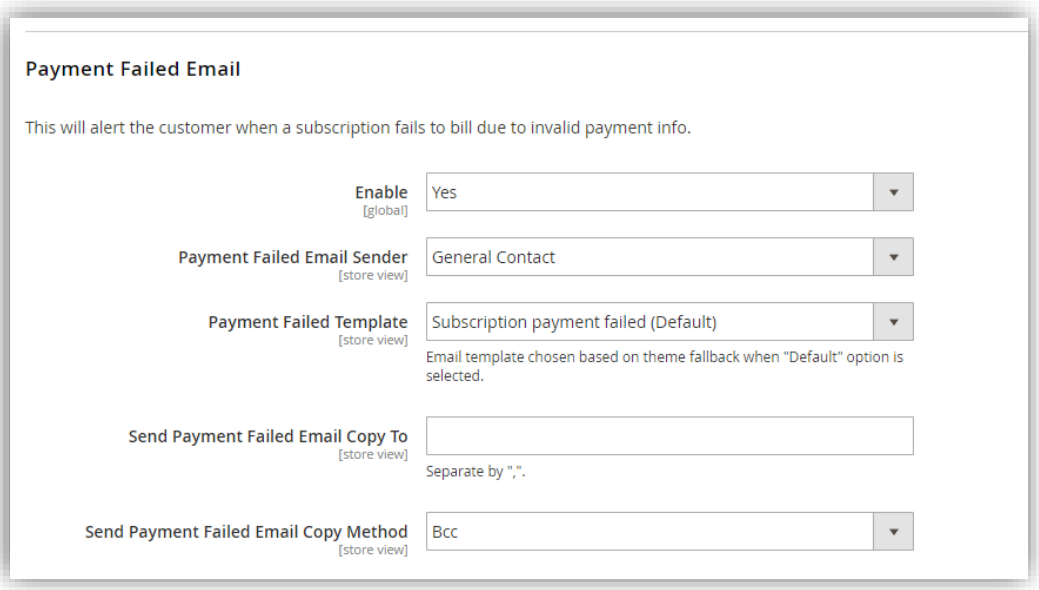
- **Enable**: If yes, an email will be sent any time a subscription fails to bill.
- **Billing Failed Email Sender**: Any billing failure emails would be sent 'from' this contact.
- **Billing Failed Email Receiver**: Any billing failure emails would be sent to this contact.
- **Billing Failed Template**: The billing failure email will use the selected template. You can customize the default template through the **Admin > Marketing > Communications > Email Templates** section.
- **Send Billing Failed Copy To**: If you want the emails to go to multiple contacts, enter the additional emails in this field, separated by commas.
- **Send Billing Failed Email Copy Method**: Choose 'CC' to include all contacts on one email; 'Bcc' to send a separate email to each.

The default email looks like:



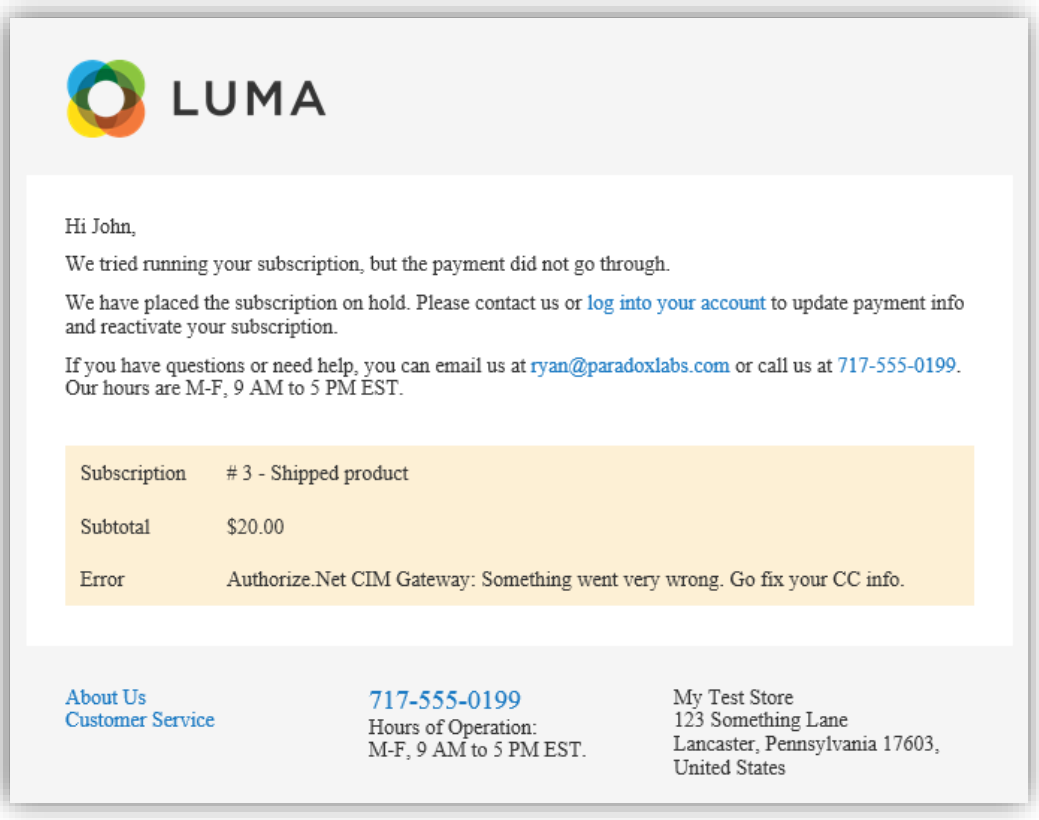
Payment Failed Email

This section allows you to control the payment failure email. This is sent to the customer if their subscription fails with a payment error (expired CC, insufficient funds, etc.). Customers will not be notified of any other failures.



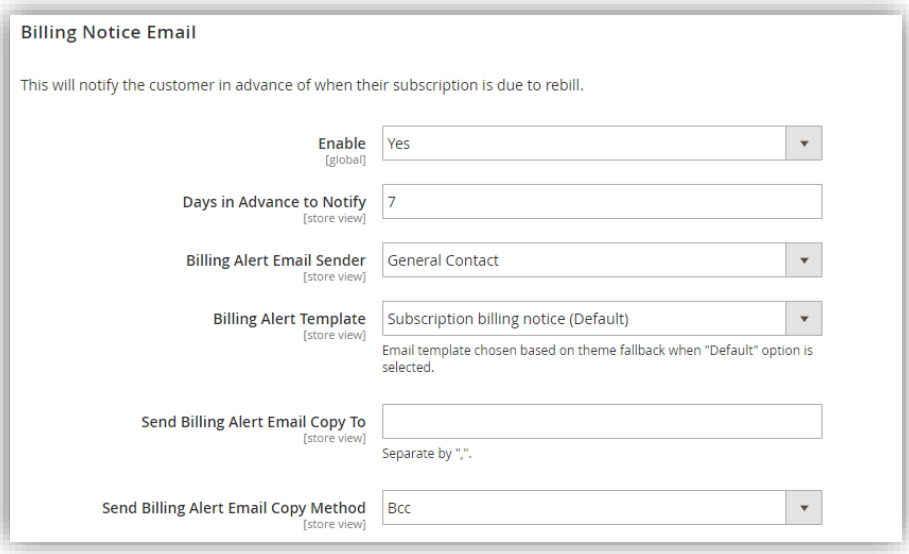
- **Enable**: If yes, the customer will be notified of subscription rebilling payment failures.
- **Payment Failed Email Sender**: Any payment failure emails would be sent 'from' this contact.
- **Payment Failed Template**: The payment failure email will use the selected template. You can customize the default template through the **Admin > Marketing > Communications > Email Templates** section.
- **Send Payment Failed Copy To**: If you want the emails to go to multiple contacts, enter the additional emails in this field, separated by commas.
- **Send Payment Failed Email Copy Method**: Choose 'CC' to include all contacts on one email; 'Bcc' to send a separate email to each.

The default email looks like:



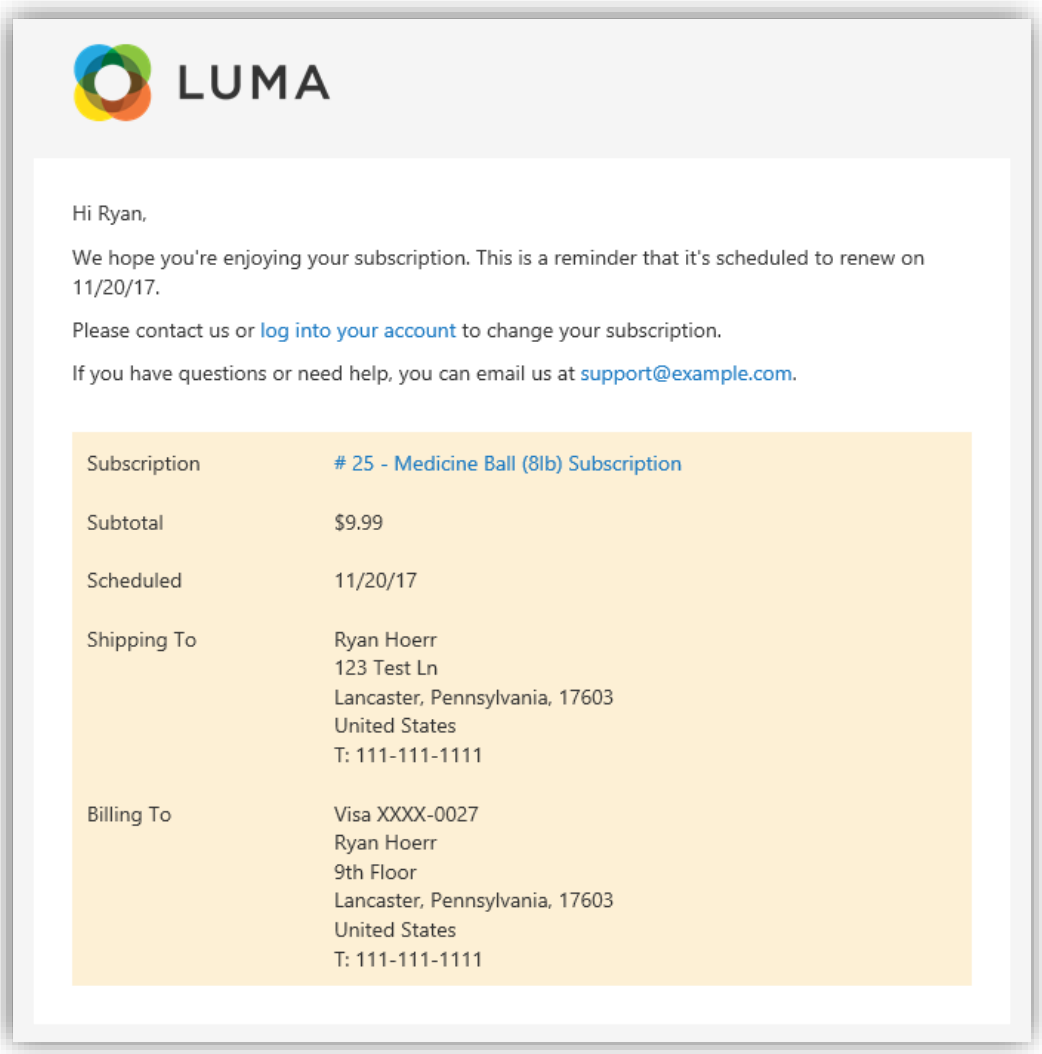
Billing Notice Email

This section allows you to control the billing notice email. This is sent to the customer in advance of their subscriptions' scheduled installment dates.



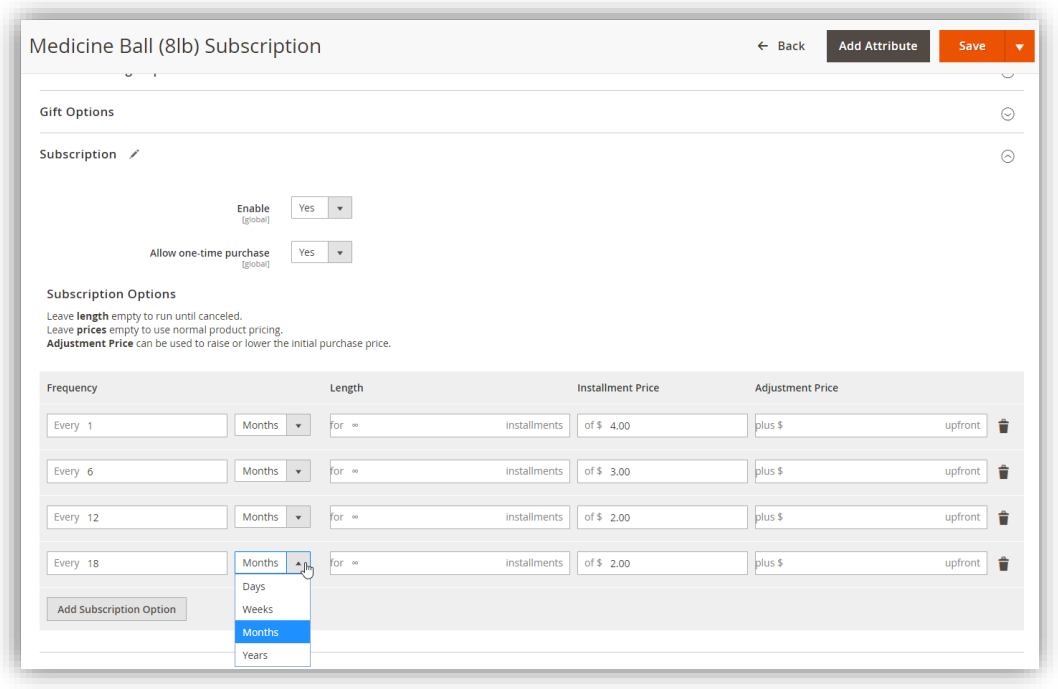
- **Enable**: If yes, the customer will be notified of upcoming subscription installments.
- **Days in Advance to Notify**: This is the number of days before the billing date that customers should be notified of an upcoming installment. The email will be sent at approximately the same time of day as they placed the original order.
- **Billing Alert Email Sender**: Emails would be sent 'from' this contact.
- **Billing Alert Template**: The email will use the selected template. You can customize the default template through the **Admin > Marketing > Communications > Email Templates** section.
- **Send Billing Alert Copy To**: If you want the emails to go to multiple contacts, enter the additional emails in this field, separated by commas.
- **Send Billing Alert Email Copy Method**: Choose 'CC' to include all contacts on one email; 'Bcc' to send a separate email to each.

The default email looks like:



Creating a Subscription Product

Once you've configured the module, you'll want to create a subscription product. Start by creating a new product, or editing the product you want to turn into a subscription, in **Admin > Products > Catalog**. On the product form, go to the **Subscriptions** section toward the bottom. You should see options like the following.



- **Enable**: This controls whether the rest of the subscription settings actually take effect. If yes, a 'subscription' custom option will be generated on save, and customers will be able to purchase the item as a subscription. If no, the item cannot be purchased as a subscription. Changing this **will not affect** any subscriptions already purchased.
  Note, **we strongly advise not disabling this once enabled**. Subscription options will not be saved for subscription-disabled products, and custom option IDs will change after reenabling, meaning this can result in data loss.
- **Allow one-time purchase**: If yes, customers will be able to choose a 'One time' option for the product, and no subscription would be generated.
- **Subscription Options**: Enter the interval options you want the customer to be able to choose from. If there is only one option, and one-time purchase is not enabled, the product will always be a subscription—customers will not have to choose a subscription interval to purchase.
  - **Frequency**: Enter the subscription interval, in conjunction with the unit dropdown (as in: Every 30 days, every 3 months, etc.).
  - **Unit**: Choose the unit for the subscription interval. Possible values: Day, week, month, year
  - **Length**: If entered (greater than 0), the subscription will run for this number of installments before ending. Otherwise, it will continue indefinitely (until failure or canceled).
  - **Installment Price**: This will override the normal product price if (and only if) the product is purchased as a subscription. The subscription price will be displayed in the cart and through

checkout. It will **not** be reflected on the product page. If no installment price is entered, the normal product price will be used.

- **Adjustment Price**: This will be added to the subscription price for the initial purchase only. The value can be positive or negative. Use this to add an additional setup fee, or give a first-installment discount. You can discount the initial purchase price all the way down to $0.00. (Note, some payment methods do not support $0 checkout.) The subscription price will be displayed in the cart and through checkout. It will **not** be reflected on the product page.

If you do not have a 'Subscription' tab, verify that the subscription attributes are in the product attribute set.

Note: For multi-option subscription products, we recommend not changing subscription options once they're set up. Adding additional options is fine, but modifying the frequency or length of existing options will generate a new custom option and remove the existing. While existing subscriptions will continue to work and bill as expected, they'll display a 'missing required options' message on the status page rather than the subscription frequency.

Product Type Compatibility

Subscription compatibility with product types depends on the features and limitations of each product type. Multiple-option subscriptions require custom options, so products that do not support custom options (like bundled products, or child-products of bundled, configurable, or grouped products) cannot have multiple options.

- **Simple products**: All subscription functionality is compatible.
- **Virtual products**: All subscription functionality is compatible.
- **Downloadable products**: All subscription functionality is compatible.
- **Configurable products**: All subscription functionality is compatible. Note that you can have multiple subscription options on the configurable product itself, but you **cannot** have multiple options on child products. You can add subscription simple or virtual products to a configurable product, as long as they do not have a custom option (only one interval).
- **Grouped products**: You cannot have group product subscriptions (you can't purchase a grouped product directly). You can add subscription simple or virtual products to a grouped product, as long as they do not have a custom option (only one interval).
- **Bundled products**: You can have single-interval bundled products; you cannot allow choice of intervals (no custom option support). Bundled products have a different form because of this limitation. You can add subscription simple or virtual products to a bundle product, as long as they do not have a custom option (only one interval).

Subscription Pricing

If the customer purchases a product as a subscription, the price will be calculated by the following logic: The installment price will override the normal product price, if any. The adjustment price (positive or negative) will be added to the installment price, for the initial purchase only.

Normal Price	Installment Price	Initial Order Adjustment	Initial Order Price	Later Installment Price
$50.00	-	-	$50.00	$50.00
$50.00	$35.00	-	$35.00	$35.00
$50.00	$35.00	$10.00	$45.00	$35.00
$50.00	$35.00	-$10.00	$25.00	$35.00
$50.00	-	-$10.00	$40.00	$50.00

You can skip the installment price, or the adjustment price, or both. An installment price will override any complex pricing the product might have (regular price, special price, grouped price, tiered price).

By default, the installment price for a subscription is calculated once when the subscription is purchased, and all installments have the same price. Changing a product's price will not affect the price of existing subscriptions for that product. This behavior can be changed by customization.

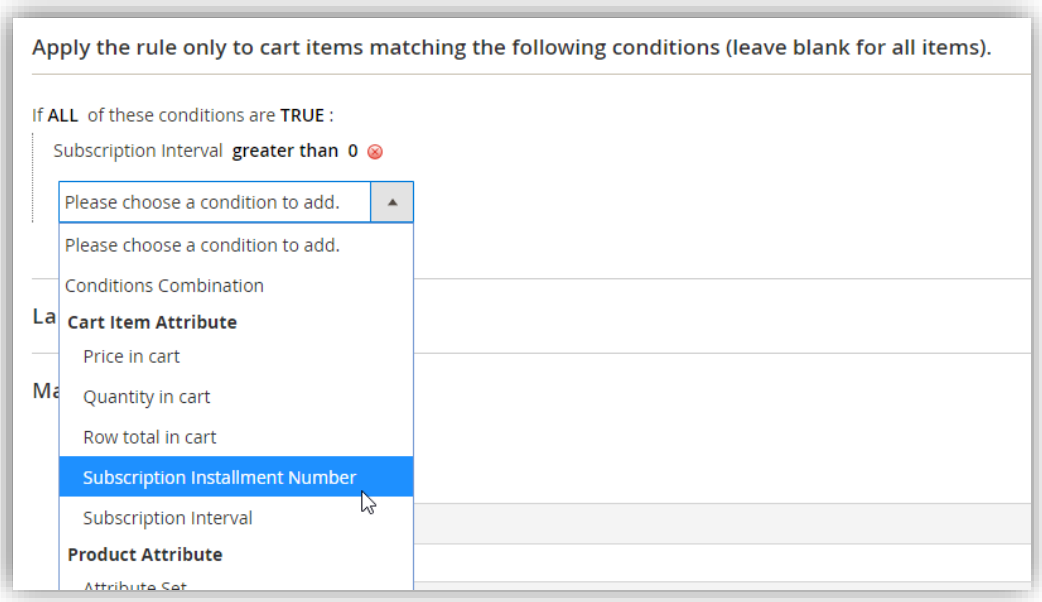
For more advanced pricing control (percentage discounts and more), consider a cart price rule (below).

Currency is automatically converted for store views according to standard Magento currency handling. All prices should be entered in your base currency.

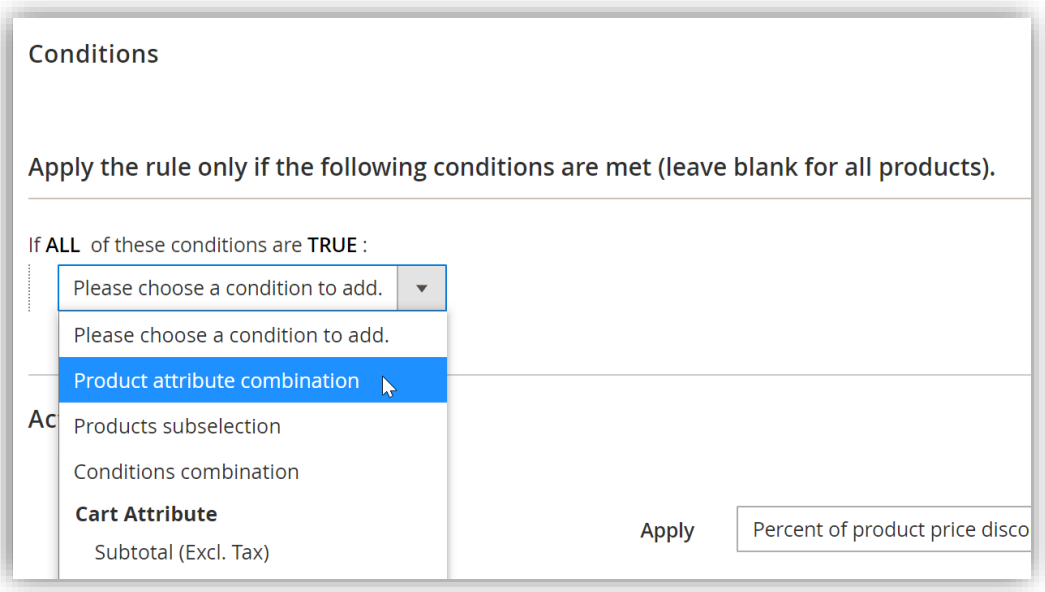
Cart Price Rules

You can also alter subscription prices using Magento's **Cart Price Rules**. You will find two new condition and action options, 'Subscription Interval' and 'Subscription Installment Number'.

- **Subscription Installment Number**: This is the installment number of the subscription being purchased. For the initial purchase, this is 1. For the second billing, the installment number is 2, for the third billing it is 3, etc.
  To apply a special discount to the initial order only, add a condition *'Subscription Installment Number equal to 1'*.
- **Subscription Interval**: This is the numeric interval the customer added to their cart. For example, if a product's interval options are 1, 2, or 3 weeks, the interval would be 1, 2, or 3. If the customer chose to purchase it one-time (with no subscription), the interval will be 0.
  To apply a special discount to subscription purchases only, add a condition *'Subscription Interval greater than 0'*.
- **Subscription Unit**: This is the frequency unit of the subscription, for instance 'day', 'week', 'month', or 'year'. Together with the subscription interval, this determines how often the subscription runs.



Note: For sales rule conditions, you must add a “Product attribute combination” or “Products subselection” condition, then you can add subscription conditions within that. Item-specific conditions are not available at the top level.



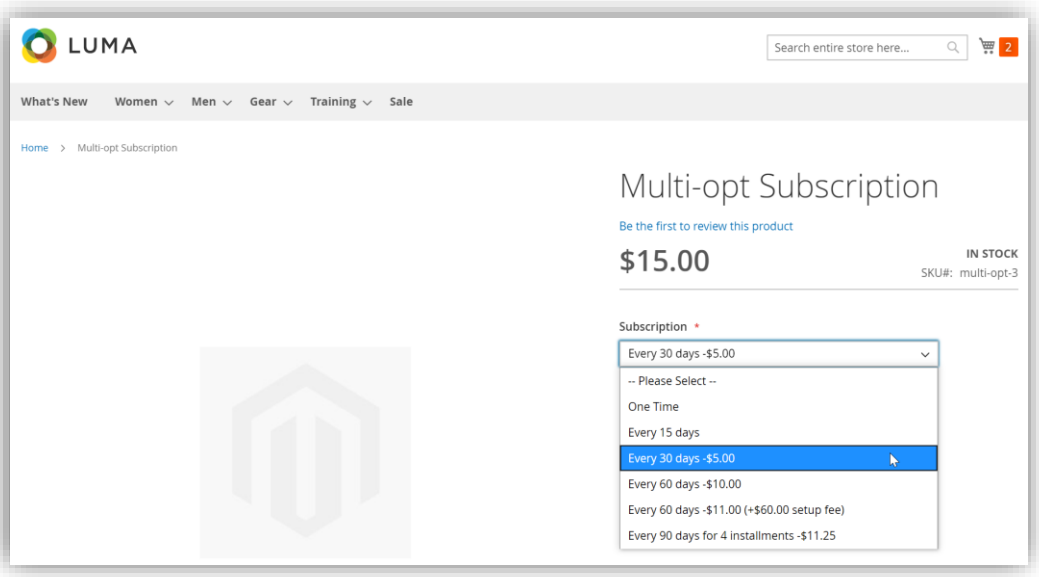
Note: Coupons *only* apply to the initial purchase of a subscription. By default, coupon codes are not copied from the original order to the subscription and later orders. That means that if you require a coupon code, a cart price rule will only apply to the initial order. You can change this behavior in the subscription extension settings.

Per standard Magento behavior, discounts from Cart Price Rules show up as a ‘Discount’ on the order totals. They do not display per-item on the frontend.

Purchasing a Subscription

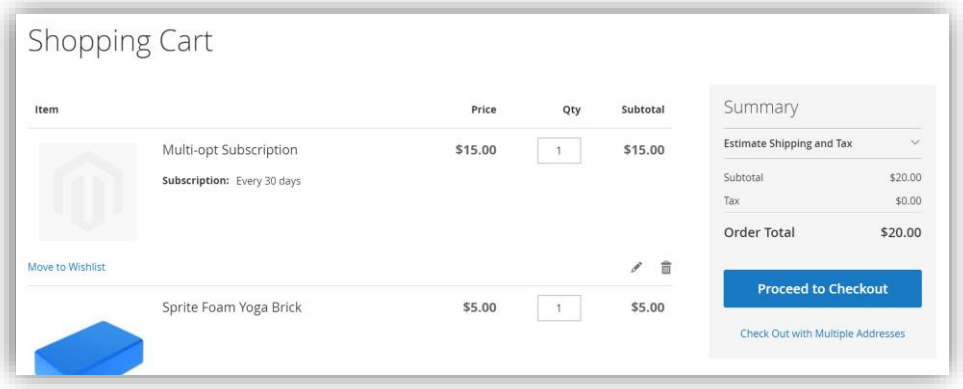
To purchase a subscription, just find it and go through the normal purchase process.

Locate your subscription product on the frontend or admin order creation. The product will have a 'subscription' option, allowing you to choose from the allowed intervals you set for that product.



Each option will automatically include its subscription pricing info, including any upfront adjustment fee or discount. The displayed product price will dynamically update to match the selected option, same as it will appear on the shopping cart.

Choose an option, then proceed through checkout. That's all there is to it. The selected option will be displayed under the product info throughout the process, and the subscription price (if any) will be shown. You can purchase any number of subscriptions and non-subscription items at the same time.



If you purchase a subscription as a guest, you will be automatically registered during checkout. An email will be sent with login instructions.

Payment Methods

Not all payment methods can be used to purchase subscriptions. Payment methods that are compatible include:

- **ParadoxLabs payment methods**: This includes **Authorize.net CIM, Carat, Clover, CyberSource**, and **Stripe**.
- **Magento Vault payment methods**: The Vault adds stored card support for some default and third-party payment methods. This includes the **Braintree** and **PayPal Payflow Pro** payment methods.
- **Third-party payment methods that support Magento Vault** should also be compatible. Some payment methods may require additional compatibility code to handle the rebilling process. Known compatible methods include **Adyen**, **Bluesnap**, and **Cayan**.
- **Offline payment methods**: This includes Bank Transfer, Cash on Delivery, Check / Money Order, Purchase Order, Zero-Total Checkout, and any third-party payment method that flags itself as 'offline'.

If you use a Vault-compatible payment method, with the Vault setting enabled, customers will be able to purchase subscriptions using that payment method. Be aware that the **Magento Vault does not provide the ability to add or update credit cards** outside of checkout.

Also note: Adaptive Subscriptions supports $0 checkout, but not all payment methods do. Of the payment methods listed above, Braintree **does not** support $0 checkout. Third-party payment methods may or may not.

For offline payment methods, the subscription edit forms will include any payment fields for that offline payment method (such as purchase order number), as well as the ability to update the billing address.

For other payment methods, management of the payment account options (stored credit cards) is entirely the responsibility of the payment method. Not all payment methods give the ability to manage stored cards outside of checkout.

Shipping Methods

All typical shipping methods should be compatible with subscriptions. Shipping cost is recalculated for each installment to account for possible rate changes over time.

All subscription installments will use the same shipping method as the original order, unless the shipping method is changed by an administrator. If the order shipping method is not available, the system will attempt to automatically fall back to the cheapest shipping method that is available.

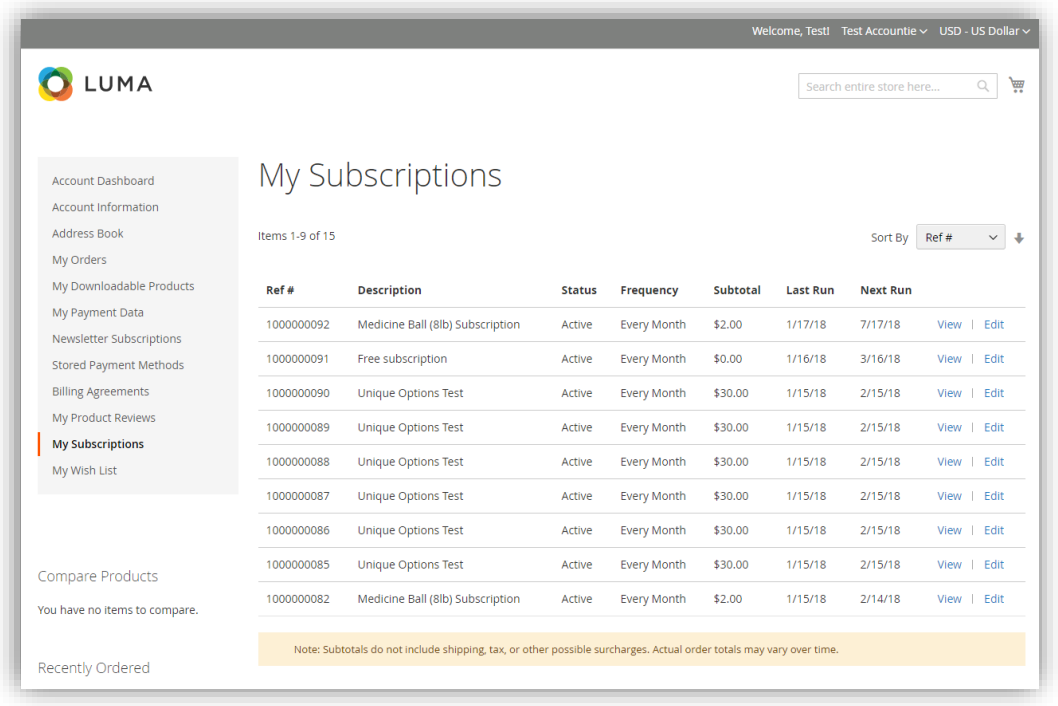
Note that free shipping rules with a minimum order total can cause unexpected behavior: Purchase of a subscription plus other items can pass the minimum for free shipping, while the subscription alone does not. If this happens, the subscription installments should fall back to the cheapest alternative shipping method.

If you wish to force all subscription installments to use a specific shipping method other than what the customer chose at checkout, there is a hidden setting that can be used to do so (`subscriptions/general/force_shipping_method`). Please contact us if needed.

Managing a Subscription

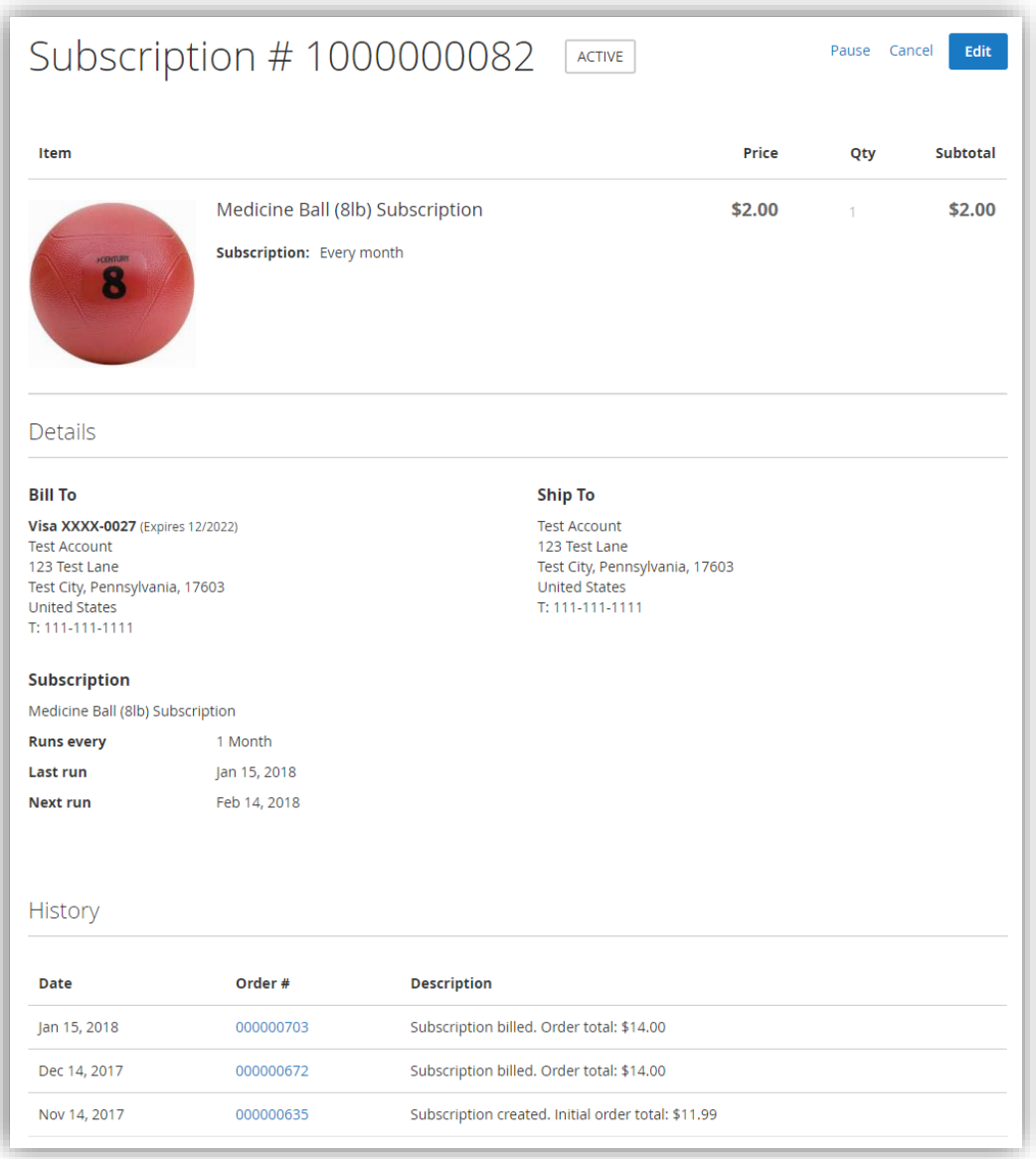
As a customer

Customers can view and manage their subscriptions from My Account, via an added 'Subscriptions' menu item on the left navigation.



Clicking 'View' will display information about the subscription item, billing and shipping addresses, payment info, frequency, and billing history.

Depending on the current subscription status and settings, the customer may have buttons allowing them to 'Pause', 'Reactivate', 'Skip', or 'Cancel'.

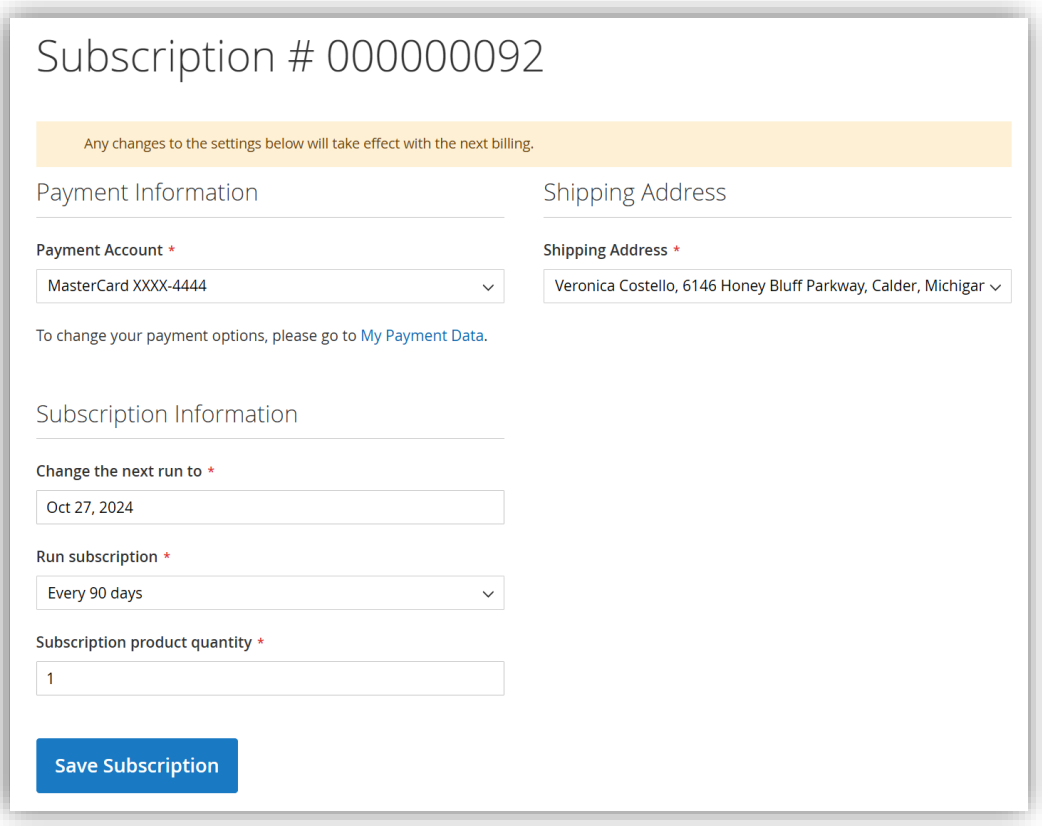


If the subscription is not complete or canceled, the customer will also be able to 'Edit'. The edit form allows the customer to change billing/payment and shipping information.

Payment account can be selected from any cards the customer has stored. The billing address is updated from that card.

Shipping address can be selected from the customer's address book, or entered manually.

If enabled, the customer can also change the subscription frequency, next run date, and product quantity. Each one must be enabled in settings.

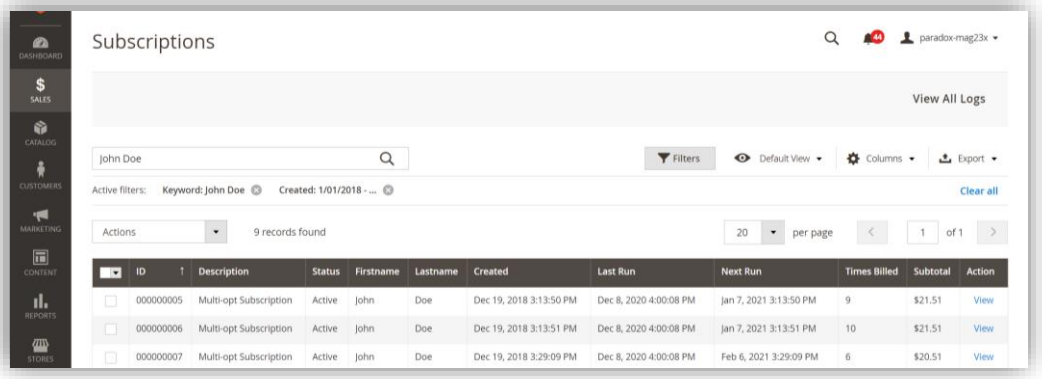


Any changes will take effect the next time the subscription runs.

As an admin

Subscriptions Grid

As administrator, you have access to some more information and options than the customer. The subscriptions grid is located at **Admin > Sales > Subscriptions > Subscriptions**.



Like other admin grids, you can customize the columns shown, sort, filter, and export the records. The keyword search field covers the customer name, email, description, subscription ID, and order IDs.

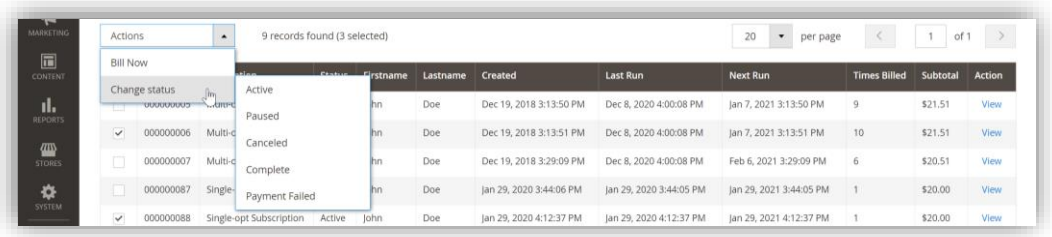
These columns are visible by default:

- ID
- Description *(subscription description—by default, the product name)*
- Status
- Firstname
- Lastname
- Created *(date)*
- Last Run *(date)*
- Next Run *(date)*
- Times Billed
- Subtotal

These additional columns are available but hidden by default. They can be enabled through the 'Columns' option at the top right, and are included in all exports:

- SKU
- Email
- Customer Group
- Updated *(date)*
- Last Notified *(date)*
- Frequency (count)
- Frequency (unit)
- Length *(number of installments)*
- Items *(number of products)*
- Qty *(total quantity of products)*
- Purchase Point *(store)*

The checkboxes on the left side allow you to select multiple subscriptions and apply a Mass Action using the dropdown at the top left:



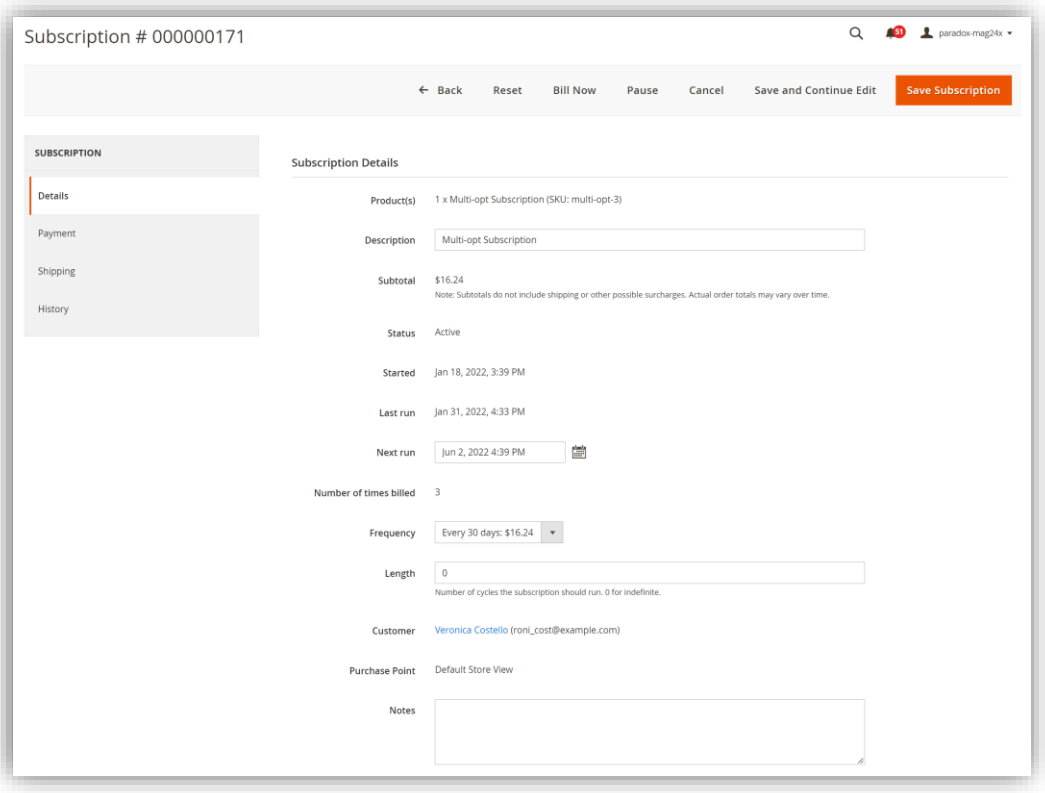
Choosing **Bill Now** will immediately renew the selected subscriptions if they are able to be billed, with no consideration of when they are next scheduled to run. Each billed subscription's next run date will be updated as if it ran normally. See *Alternate ways to run billing* for more information.

Choosing **Change Status** will change the selected subscriptions to the new status. This will only apply to subscriptions that are eligible to be changed to the new status. For instance, active subscriptions can be changed to paused, and vice versa. Canceled and complete subscriptions cannot be changed back to active.

Any mass action billing or status changes will be recorded in the subscription history logs.

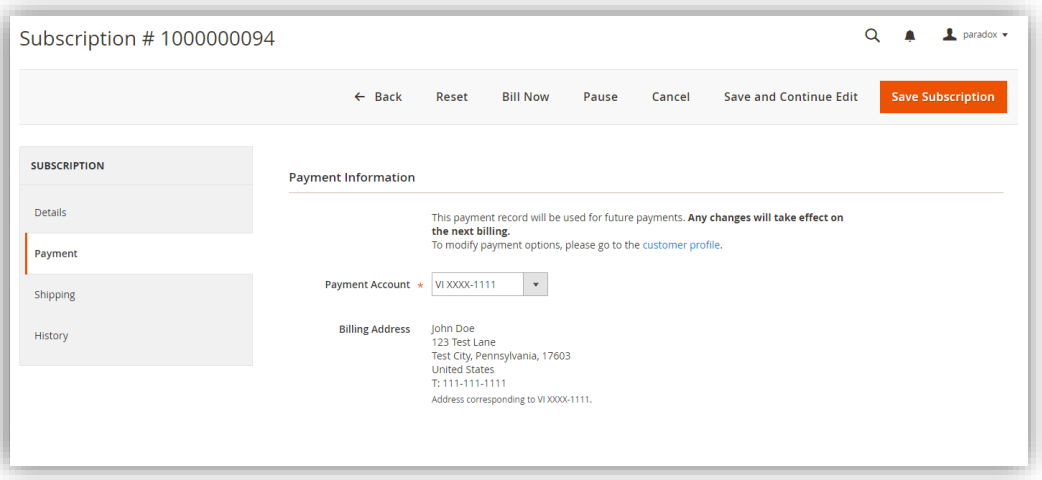
Subscription Edit

Click Edit on a subscription to get the full information and edit form.

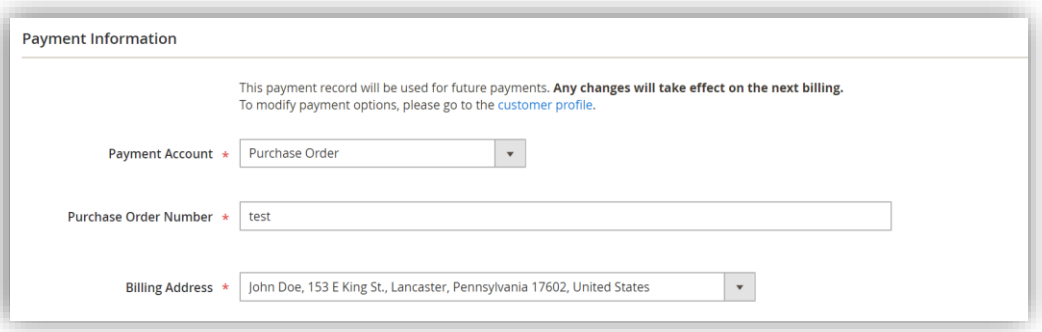


The top buttons allow you to change the subscription status, based on the current status and available options.

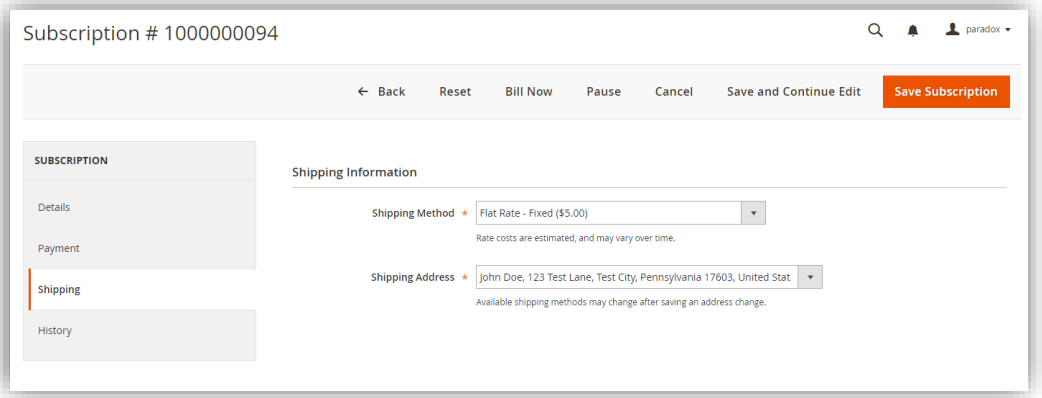
The form allows you to change the subscription description, product quantity, next run (installment) date, frequency, and length. These options are not given to the customer directly. You can also add reference notes for internal use—these will only be visible to administrators.



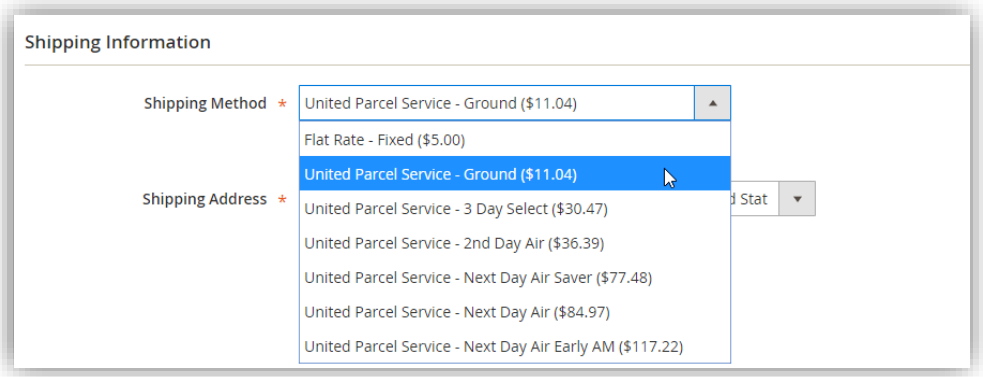
The payment tab allows you to change the payment account, from the customer's saved cards. You'll need to go to the customer's profile to add or edit cards before assigning to the subscription (ParadoxLabs payment methods only).



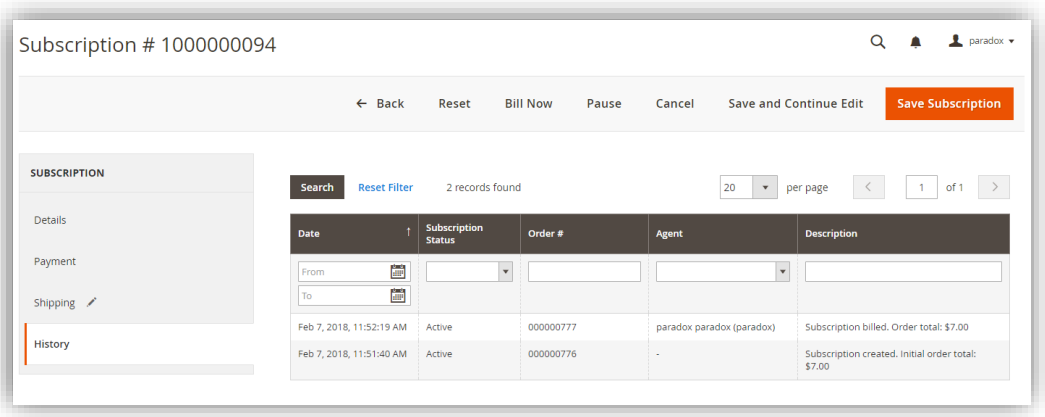
If an offline payment method is selected, the form will change to include billing address and the payment method's fields (if any).



The shipping tab allows you to change the shipping method and shipping address of the subscription. You can choose from the customer's stored addresses, or enter an address manually.



The available shipping methods and rates are based on the shipping address currently saved. If you change the shipping address, you will need to save the changes to see the rates for it. Rates for live shipping methods (UPS, USPS, FedEx, etc.) may change over time.

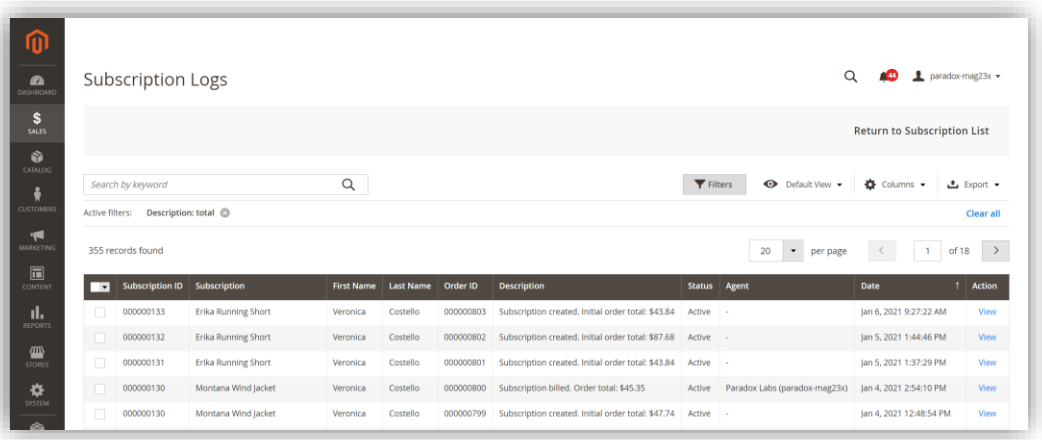


Finally, the history tab shows you everything that has happened with the subscription: Every billing, every failure, and every change of status or info. Note that customers will only see billing logs on the frontend.

Billing logs include the order number and total, and allow you to click through to the full order details.

Recent Activity

You can also view subscription logs (history) for all subscriptions at once, by going to **Admin > Sales > Subscriptions > Recent Activity**. Use this to track billings, catch and reconcile failures, see what subscription changes customers are making, and more.



Like other admin grids, you can customize the columns shown, sort, filter, and export the records. On the log grid, the keyword search field covers the order ID, status, and description.

These columns are visible by default:

- Subscription ID
- Subscription
- First Name
- Last Name
- Order ID
- Description *(log message)*
- Status
- Agent
- Date

These additional columns are available but hidden by default. They can be enabled through the ‘Columns’ option at the top right, and are included in all exports:

- Email
- Next Run *(date—note: current data, not historic)*
- Purchase Point *(store)*

Billing and Scheduling

By default, with scheduled billing enabled and Magento's cron set up, the billing process will run every hour. Each time it runs, it looks for any subscriptions with a 'next run' date in the past, and then generates an order for them.

Disable scheduled billing in configuration to prevent subscriptions from running automatically. We strongly recommend that you do this in any non-production environments to prevent any chance of accidental double-billing.

Subscription grouping

If you have the *'Group same-day subscriptions'* setting enabled, the billing process will pull all subscriptions due through midnight of the day it runs, and group them together by fulfillment info (store, customer, payment record, shipping address, shipping method). Each resulting group of subscriptions will be billed together if any one of the subscriptions in that group is due. The net result is that all of the items from all of the subscriptions in that group will be combined into one single order, with one single shipping charge.

If any of the products in a group is disabled, out of stock, or otherwise unavailable, that product's subscription will be paused and removed from the billing group. The rest of the group (if any) will continue processing as normal.

Schedule calculation

Any time a subscription runs successfully, the 'next run' date (next installment) is calculated. This is always calculated forward from the last 'next run' date, meaning subscriptions will always follow their original schedule unless that 'next run' date explicitly changed.

For example, say a 1-month subscription was purchased March 15th, and so has a next run date of April 15th. But you run it ahead of time, on April 9th. The next run date after that would be calculated as April 15th + 1 month = May 15th – *not* May 9th (1 month from the date billed). Or, alternatively, your Magento cron malfunctions, and the subscription that was due April 15th doesn't get billed until the 18th. Again, the next run date would be calculated as April 15th + 1 month = May 15th – *not* May 18th. If desired, this behavior is easily customized in code.

Schedule blackouts

Global settings allow you to restrict subscriptions to only bill on certain days of week, days of month, or months of year, and to specifically exclude additional blackout dates (like holidays). You'll find those settings in global Magento configuration (see Scheduling).

All billing calculations are subject to those scheduling blackout settings. Scheduling will choose the next available day after the expected run date. Following the above example, if you only allow billing on weekdays and May 15th is a Saturday, the next run date would be set to May 17th (Monday). If May 17th was set as a blackout date (say it's a holiday), the date would be May 18th instead. The following installment would then be on May 18th + 1 month = June 18th, assuming that is a weekday.

Any changes you make to the scheduling blackout settings will not affect existing scheduled run dates (schedule changes are not retroactive). They would take effect as each subscription bills and the next run date gets calculated.

Regardless of your scheduling blackout settings, billing is dependent on your billing process. If you disable automatic scheduled billing, and do not have anything else in place to run them on a schedule, they will only ever bill when you manually trigger them.

Alternate ways to run billing

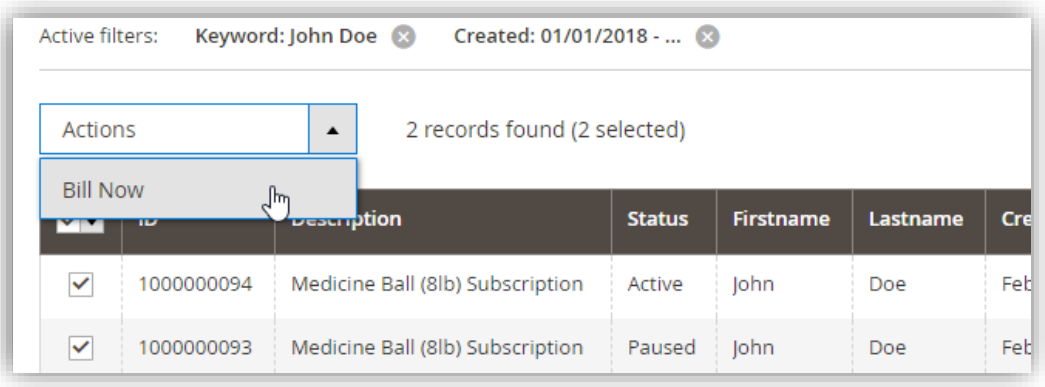
There are other ways you can run billing, in addition to the scheduled process.

Command line: We've included a 'bill' process in Magento's command-line interface. This runs the exact same process as the scheduled cron task, but with several benefits: (1) It will not interfere with any other Magento scheduled tasks; (2) you can control when it runs; and (3) you can control where it runs from (if you have a multi-server environment). Run it by executing the following command via SSH, from the site root:

```
php bin/magento subscriptions:bill
```

You could use this in a number of ways, including automated billing on a more limited basis than the normal scheduled task. For instance, you could set it as a cron job to run the 1st and 15th of every month, rather than every hour; or to only run overnight.

Manual billing: The admin subscriptions grid allows you to check particular subscriptions and mass-bill. This will generate an order for each of the subscriptions in question, regardless of whether those subscriptions were due or not.



You can use this to bill certain subscriptions ahead of schedule (say a customer calls in and wants to receive the product before a vacation), or to run all of your subscriptions under total manual control.

You will find this useful if you want complete control over exactly when subscriptions run. For example, If you have a club product that runs once every month, you may want to bill and fulfill all of them together, and you may not want to do that until you actually have the product ready. You can get everything ready, wait until the first business day of the month or whatever your schedule might be, and then select all of the relevant subscriptions and generate their orders all at once.

There is also a 'Bill Now' button on the edit page, which behaves the same way.

Error Handling

Any error of any sort during the order generation process (rebilling a subscription) will place the subscription on hold. If the failure is payment-related (invalid/expired credit card, insufficient funds, etc.), the subscription will be moved to 'Payment Failed' status. For any other error (product disabled or out of stock, API failure, etc.), the subscription will be moved to 'Paused' status. Either way, the subscription will be on hold indefinitely until reactivated by customer or admin.

For payment failures, the customer will be sent a notification email, subject to the 'Payment Failed Email' settings discussed in Configuration.

For all subscription failures (including payment failures), the store owner will be sent a notification email, subject to the 'Billing Failed Email' settings. The customer will not be notified except for payment failures.

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.

Technical / Integration Details

Architecture

Products are defined as subscriptions via added product attributes and metadata. The 'subscription options' grid is stored in `paradoxlabs_subscription_product_interval`, model `\ParadoxLabs\Subscription\Model\Interval`. Each subscription option becomes a custom option value on the product, and each custom option value has a corresponding Interval storing the product, option, and value IDs, and the additional subscription option data (frequency, prices, etc.). All of that is generated dynamically upon product save. If there is only one option for the product (no custom option), the values will be stored on the product subscription attributes and there will be no `Interval`.

Any custom subscription price is applied as `custom_option_price` when added to the cart. On purchase, after order place, any subscription items are converted into actual subscriptions (one per item) during the `sales_model_service_quote_submit_success` event.

Those subscriptions are stored in `paradoxlabs_subscription`, model `\ParadoxLabs\Subscriptions\Model\Subscription`, containing data on run schedule, length, customer, etc. The entirety of the fulfillment information (billing, payment, shipping, items) is stored in a quote, tied by column `quote_id`. On each billing, the storage quote is duplicated and then converted to an order.

Billing happens in `\ParadoxLabs\Subscriptions\Model\Service\Subscription::generateOrder()`, by various entry points (admin interface, command line interface, crontab scheduled task).

Every change and occurrence regarding a subscription is recorded in table `paradoxlabs_subscription_log`, model `\ParadoxLabs\Subscriptions\Model\Log`.

If you are planning customizations: Start by exploring the `\ParadoxLabs\Subscriptions\Model\Service` folder. Use events or plugins wherever possible. If you must rewrite a class, extend the class and copy/replace as little code as possible to accomplish your goal. Events will be most resistant to breaking changes from future updates. Please never modify our code directly unless instructed to do so.

Custom product attributes

- subscription_active
- subscription_allow_onetime
- subscription_intervals
- subscription_unit
- subscription_length
- subscription_price
- subscription_init_adjustment

Custom database tables

- paradoxlabs_subscription
- paradoxlabs_subscription_log
- paradoxlabs_subscription_product_interval
- sequence_subscription_*

Events

- `paradoxlabs_subscription_collect_totals_before` (`quote`, `subscriptions`): Fires for each subscription billing occurrence, after the quote is generated, but before totals or shipping rates are collected. Use this to alter prices or make other pre-billing adjustments.
- `paradoxlabs_subscription_generate_before` (`quote`, `subscriptions`): Fires for each subscription billing occurrence, after the quote is generated, before being converted to an order. Use this to alter subscription orders before they go through.
- `paradoxlabs_subscription_prepare_payment_{paymentMethodCode}` (`quote`, `subscriptions`): Fires immediately after the `generate_before` event, before the subscription billing occurs. Use this for payment method initialization before rebilling. The dynamic event code allows soft dependencies
- `paradoxlabs_subscription_generate_after` (`quote`, `order`, `subscriptions`): Fires for each subscription billing occurrence, after order has been created and placed successfully, but before the subscription(s) are saved and order email is sent. Use this to perform follow-up actions after a subscription billing.
- `paradoxlabs_subscription_generate_save_after` (`quote`, `order`, `subscriptions`): Fires for each subscription billing occurrence, after the order has been created and placed successfully, and after the subscription, quote, and order have all saved. Use this to perform any follow-up actions that require having an order ID.
- `paradoxlabs_subscription_status_change` (`old_status`, `new_status`, `message`, `subscription`): Fires each time a subscription status changes, in any way. Use this to perform actions related to changed subscription status (active, paused, payment_failed, completed, canceled, etc.).
- `paradoxlabs_subscription_billing_failed` (`subscriptions`, `exception`): Fires for each subscription billing occurrence that fails, prior to the status being changed or failure emails being sent.
- `adminhtml_subscription_view_tab_main_prepare_form` (`form`): Fires after adding standard fields to the admin subscription edit form, 'main' tab. Use this to add or modify fields on the subscription form.
- `adminhtml_subscription_view_tab_payment_prepare_form` (`form`): Fires after adding standard fields to the admin subscription edit form, 'payment' tab.
- `paradoxlabs_subscription_interval_duplicate_before` (`old_interval`, `new_interval`, `new_product_option`): Fires after cloning each Interval when duplicating a product, before the interval is saved.
- `paradoxlabs_subscription_interval_duplicate_after` (`old_interval`, `new_interval`, `new_product_option`): Fires after cloning each Interval when duplicating a product, after the interval is saved.
- `paradoxlabs_subscription_billing_failed_set_email_vars` (`sender`, `transport`): Fires before sending the admin Billing Failed email, allowing customization of the email template variables (in `transport`).
- `paradoxlabs_subscription_payment_failed_set_email_vars` (`sender`, `transport`): Fires before sending the Payment Failed email, allowing customization of the email template variables (in `transport`).
- `paradoxlabs_subscription_billing_notice_set_email_vars` (`sender`, `transport`): Fires before sending the customer upcoming-billing notification email, allowing customization of the email template variables (in `transport`).
- `paradoxlabs_subscription_billing_hash_fulfillment_info` (`subscription`, `service`, `transport`): Fires during subscription hash calculation during rebilling, allowing customization of the subscription grouping logic.
- `paradoxlabs_new_subscription_save_before` (`subscription`, `source_order`, `source_item`): Fires during order submission when a new subscription is purchased, before the new subscription is saved.
- `paradoxlabs_new_subscription_collect_totals_before` (`quote`, `source_order`, `source_item`): Fires during new subscription generation, before totals are collected for the new subscription quote.

Magento API: REST and SOAP

This module supports the Magento API via standard interfaces. You can use it to create, read, update, and delete subscriptions and subscription logs, and trigger billings.

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

You can purchase/generate a subscription by creating an order for the associated product(s), with the necessary option selected. See the Magento REST API Swagger documentation for how to fetch a product's custom options and select them when adding a product to cart.

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

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/subscription/:subscriptionId (get subscription by ID)
Example request:

```
GET /rest/V1/subscription/42 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 42,
    "quote_id": 428,
    "description": "Medicine Ball (8lb) Subscription",
    "customer_id": 1,
    "created_at": "2017-09-11 15:18:41",
    "updated_at": "2017-09-11 15:21:29",
    "store_id": 1,
    "next_run": "2017-11-11 15:18:41",
    "last_run": "2017-09-11 15:21:29",
    "subtotal": 9.99,
    "complete": false,
    "length": 0,
    "run_count": 2,
    "status": "active",
    "frequency_count": 1,
    "frequency_unit": "month",
    "additional_information": []
}
```

GET /V1/subscription/search (get subscription(s) by searchCriteria)
Example request:

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

Example response:

```
{
    "items": [
        {
            "id": 1,
            // … other subscription data
        }
    ],
    "search_criteria": {
```

```
        "filter_groups": [],
        "page_size": 1
    },
    "total_count": 42
}
```

See also: Search using REST APIs (Magento DevDocs)

POST /V1/subscription (create subscription)

Example request:

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

{
    "subscription": {
        "quote_id": 428,
        "description": "Medicine Ball (8lb) Subscription",
        "customer_id": 1,
        "created_at": "2017-09-11 15:18:41",
        "updated_at": "2017-09-11 15:21:29",
        "store_id": 1,
        "next_run": "2017-11-11 15:18:41",
        "last_run": "2017-09-11 15:21:29",
        "subtotal": 9.99,
        "complete": false,
        "length": 0,
        "run_count": 2,
        "status": "active",
        "frequency_count": 1,
        "frequency_unit": "month",
        "additional_information": []
    }
}
```

Example response:

```
{
    "id": 43,
    "quote_id": 428,
    "description": "Medicine Ball (8lb) Subscription",
    "customer_id": 1,
    "created_at": "2017-09-25 18:26:37",
    "updated_at": "2017-09-25 18:26:37",
    "store_id": 1,
    "next_run": "2017-11-11 15:18:41",
    "last_run": "2017-09-11 15:21:29",
    "subtotal": 9.99,
    "complete": false,
    "length": 0,
    "run_count": 2,
    "status": "active",
    "frequency_count": 1,
    "frequency_unit": "month",
    "additional_information": []
}
```

PUT /V1/subscription/:subscriptionId (update subscription)

Example request:

```
PUT /rest/V1/subscription/43 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "subscription": {
        "id": 43,
        "quote_id": 428,
        "description": "Medicine Ball (8lb) Subscription",
        "customer_id": 1,
```

```
        "created_at": "2017-09-11 15:18:41",
        "updated_at": "2017-09-11 15:21:29",
        "store_id": 1,
        "next_run": "2017-11-11 15:18:41",
        "last_run": "2017-09-11 15:21:29",
        "subtotal": 9.99,
        "complete": false,
        "length": 0,
        "run_count": 2,
        "status": "active",
        "frequency_count": 1,
        "frequency_unit": "month",
        "additional_information": []
    }
}
```

Example response:

```
{
    "id": 43,
    "quote_id": 428,
    "description": "Medicine Ball (8lb) Subscription",
    "customer_id": 1,
    "created_at": "2017-09-11 15:18:41",
    "updated_at": "2017-09-25 18:27:52",
    "store_id": 1,
    "next_run": "2017-11-11 15:18:41",
    "last_run": "2017-09-11 15:21:29",
    "subtotal": 9.99,
    "complete": false,
    "length": 0,
    "run_count": 2,
    "status": "active",
    "frequency_count": 1,
    "frequency_unit": "month",
    "additional_information": []
}
```

POST /V1/subscription/:subscriptionId/bill (bill subscription by ID)

Example request:

```
POST /rest/V1/subscription/43/bill HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
true
```

DELETE /V1/subscription/:subscriptionId (delete subscription by ID)

Example request:

```
DELETE /rest/V1/subscription/43 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
true
```

GET /V1/subscriptionLog/:logId (get log by ID)

Example request:

```
GET /rest/V1/subscriptionLog/234 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 234,
    "subscription_id": 42,
    "status": "active",
    "order_increment_id": "000000505",
    "order_id": "268",
    "description": "Subscription billed. Order total: $9.99",
    "agent_id": 1,
    "created_at": "2017-09-11 15:21:29",
    "additional_information": []
}
```

GET /V1/subscriptionLog/search (get log(s) by searchCriteria)

Example request:

```
GET /rest/V1/subscriptionLog/search
    ?searchCriteria[filter_groups][0][filters][0][field]=subscription_id
    &searchCriteria[filter_groups][0][filters][0][value]=42 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "items": [
        {
            "id": 233,
            "subscription_id": 42,
            "status": "active",
            "order_increment_id": "000000504",
            "order_id": null,
            "description": "Subscription created. Initial order total: $9.99",
            "agent_id": 0,
            "created_at": "2017-09-11 15:18:41",
            "additional_information": []
        },
        {
            "id": 234,
            "subscription_id": 42,
            "status": "active",
            "order_increment_id": "000000505",
            "order_id": "268",
            "description": "Subscription billed. Order total: $9.99",
            "agent_id": 1,
            "created_at": "2017-09-11 15:21:29",
            "additional_information": []
        }
    ],
    "search_criteria": {
        "filter_groups": [
            {
                "filters": [
                    {
                        "field": "subscription_id",
                        "value": "42",
                        "condition_type": "eq"
                    }
                ]
            }
        ]
    },
    "total_count": 2
}
```

POST /V1/subscriptionLog (create log)

Example request:

```
POST /rest/V1/subscriptionLog HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

```
Content-Type: application/json

{
    "log": {
        "subscription_id": 42,
        "status": "active",
        "order_increment_id": null,
        "order_id": null,
        "description": "Custom subscription log",
        "agent_id": 0,
        "additional_information": []
    }
}
```

Example response:

```
{
    "id": 238,
    "subscription_id": 42,
    "status": "active",
    "order_increment_id": null,
    "order_id": null,
    "description": "Custom subscription log",
    "agent_id": 0,
    "created_at": "2017-09-25 18:37:12",
    "additional_information": []
}
```

PUT /V1/subscriptionLog/:logId (update log)

Example request:

```
PUT /rest/V1/subscriptionLog/238 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json

{
    "log": {
        "id": 238,
        "subscription_id": 42,
        "status": "active",
        "order_increment_id": "",
        "order_id": "",
        "description": "Custom subscription log",
        "agent_id": 0,
        "created_at": "2017-09-25 18:37:12",
        "additional_information": []
    }
}
```

Example response:

```
{
    "id": 238,
    "subscription_id": 42,
    "status": "active",
    "order_increment_id": "",
    "order_id": "",
    "description": "Custom subscription log",
    "agent_id": 0,
    "created_at": "2017-09-25 18:37:12",
    "additional_information": []
}
```

DELETE /V1/subscriptionLog/:logId (delete log by ID)

Example request:

```
DELETE /rest/V1/subscriptionLog/238 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
true
```

GET /V1/subscriptionInterval/:intervalId (get product subscription interval by ID)
GET /V1/subscriptionInterval/search (get product subscription intervals by searchCriteria)
GET /V1/subscriptionInterval/product/:productid (get product subscription intervals by product ID)
GET /V1/subscriptionInterval/option/:optionId (get product subscription intervals by option ID)
POST /V1/subscriptionInterval (create product subscription interval)
PUT /V1/subscriptionInterval/:intervalId (update product subscription interval)
DELETE /V1/subscriptionInterval/:intervalId (delete product subscription interval)
Note: These subscription interval endpoints are exposed for advanced users. Intervals directly relate to a product custom option value by definition. They are created automatically when saving a subscription-enabled product with multiple options. We expose them by API so that the data is accessible, and because this may allow customization in ways we don't necessarily foresee.

Creating intervals manually would require also creating the associated custom option values manually, in advance, in order for everything to work as expected.

Please see Swagger documentation if you need specifics on usage of these endpoints.

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 > Catalog > Adaptive Subscriptions > Enable public API**. Only enable this if you use them.

GET /V1/subscription/mine/:subscriptionId (Get customer's subscription by ID)
Example request:

```
GET /rest/V1/subscription/mine/2 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:

```
{
    "id": 2,
    "increment_id": "000000002",
    "quote_id": 5,
    "description": "Multi-opt Subscription",
    "customer_id": 2,
    "created_at": "2018-11-28 17:28:40",
    "updated_at": "2019-03-08 18:57:58",
    "store_id": 1,
    "next_run": "2019-01-27 17:28:40",
    "last_run": "2018-11-28 17:28:40",
    "last_notified": "0000-00-00 00:00:00",
    "subtotal": 18.01,
    "complete": false,
    "length": 0,
    "run_count": 1,
    "status": "active",
    "frequency_count": 60,
    "frequency_unit": "day",
```

```
    "additional_information": [],
    "keyword_fulltext": "John Doe email@example.com 000000002 000000003"
}
```

GET /V1/subscription/mine/search (Search customer's subscriptions)

Example request:

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

Example response:

```
{
    "items": [
        ...
    ],
    "search_criteria": {
        "filter_groups": [
            {
                "filters": [
                    {
                        "field": "customer_id",
                        "value": "2",
                        "condition_type": "eq"
                    }
                ]
            }
        ],
        "page_size": 1
    },
    "total_count": 4
}
```

See also: Search using REST APIs (Magento DevDocs)

POST /V1/subscription/mine (Create subscription for customer)

Example request:

```
POST /rest/V1/subscription/mine HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json
{
  "subscription": {
      "quote_id": 9,
      "description": "Medicine Ball (8lb) Subscription",
      "customer_id": 2,
      "created_at": "2017-09-11 15:18:41",
      "updated_at": "2017-09-11 15:21:29",
      "store_id": 1,
      "next_run": "2017-11-11 15:18:41",
      "last_run": "2017-09-11 15:21:29",
      "subtotal": 9.99,
      "complete": false,
      "length": 0,
      "run_count": 2,
      "status": "active",
      "frequency_count": 1,
      "frequency_unit": "month",
      "additional_information": []
  }
}
```

Example response:

```
{
    "id": 18,
    "increment_id": "000000018",
    "quote_id": 9,
    "description": "Medicine Ball (8lb) Subscription",
```

```
    "customer_id": 2,
    "created_at": "2019-04-25 03:11:28",
    "updated_at": "2019-04-25 03:11:28",
    "store_id": 1,
    "next_run": "2017-11-11 15:18:41",
    "last_run": "2017-09-11 15:21:29",
    "last_notified": null,
    "subtotal": 9.99,
    "complete": false,
    "length": 0,
    "run_count": 2,
    "status": "active",
    "frequency_count": 1,
    "frequency_unit": "month",
    "additional_information": [],
    "keyword_fulltext": "John Doe email@example.com 000000018"
}
```


PUT /V1/subscription/mine/:subscriptionId (Update subscription for customer)

Example request:


```
PUT /rest/V1/subscription/mine/2 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
Content-Type: application/json
{
    "subscription": {
            "id": 2,
            "increment_id": "000000002",
            "quote_id": 5,
            "description": "Multi-opt Subscription",
            "customer_id": 2,
            "created_at": "2018-11-28 17:28:40",
            "updated_at": "2018-11-28 17:28:40",
            "store_id": 1,
            "next_run": "2019-01-27 17:28:40",
            "last_run": "2018-11-28 17:28:40",
            "last_notified": "0000-00-00 00:00:00",
            "subtotal": 18.01,
            "complete": false,
            "length": 0,
            "run_count": 1,
            "status": "active",
            "frequency_count": 60,
            "frequency_unit": "day",
            "additional_information": [],
            "keyword_fulltext": "John Doe email@example.com 000000002 000000003"
        }
}
```


Example response:


```
{
    "id": 2,
    "increment_id": "000000002",
    "quote_id": 5,
    "description": "Multi-opt Subscription",
    "customer_id": 2,
    "created_at": "2018-11-28 17:28:40",
    "updated_at": "2019-04-25 03:12:33",
    "store_id": 1,
    "next_run": "2019-01-27 17:28:40",
    "last_run": "2018-11-28 17:28:40",
    "last_notified": "-0001-11-30 00:00:00",
    "subtotal": 18.01,
    "complete": false,
    "length": 0,
    "run_count": 1,
    "status": "active",
    "frequency_count": 60,
    "frequency_unit": "day",
    "additional_information": [],
    "keyword_fulltext": "John Doe email@example.com 000000002 000000003"
}
```

GET /V1/subscriptionLog/mine/search (Get customer's subscription logs)

Example request:

```
GET /rest/V1/subscriptionLog/mine/search?searchCriteria[pageSize]=2 HTTP/1.1
Host: {host}
Authorization: Bearer {api_key}
```

Example response:


```
{
    "items": [
        {
            "id": 1,
            "subscription_id": 1,
            "status": "active",
            "order_increment_id": "000000003",
            "order_id": "3",
            "description": "Subscription created. Initial order total: $108.01",
            "agent_id": 0,
            "created_at": "2018-11-28 17:28:40",
            "additional_information": []
        },
        …
    ],
    "search_criteria": {
        "filter_groups": [
            {
                "filters": [
                    {
                        "field": "customer_id",
                        "value": "2",
                        "condition_type": "eq"
                    }
                ]
            },
            {
                "filters": [
                    {
                        "field": "order_increment_id",
                        "value": "1",
                        "condition_type": "notnull"
                    }
                ]
            }
        ],
        "page_size": 2
    },
    "total_count": 5
}
```


See also: Search using REST APIs (Magento DevDocs)

Magento API: GraphQL

For Magento 2.3.1+, this extension supports the GraphQL API for subscription management. This is intended for PWA and headless implementations where subscription management needs to be exposed outside of Magento's standard frontend.

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

Note: These requests are disabled by default. You can enable them at **Admin > Stores > Configuration > Catalog > Adaptive Subscriptions > 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

subscriptions(entity_id: Int): [Subscription]
Get the current customer's subscriptions, if any. Takes an entity_id for a specific subscription (optional); returns one or more `Subscription` records.

subscriptionPayments(subscription_id: Int): [SubscriptionPaymentAccount]
Get the current customer's payment accounts available for a specific subscription, if any. This will provide the same payment options available in the "Payment Account" dropdown when editing a subscription through the standard frontend (all active cards the customer has from all compatible payment methods).

Subscription ID is required in order to check payment method restrictions (min/max amount, billing country, etc.). Returns zero or more `SubscriptionPaymentAccount` records.

Mutations

changeSubscriptionStatus(entity_id: Int!, status: String!): Subscription
Change the subscription matching `entity_id` to `status`. Returns the `Subscription` if successful.

updateSubscription(input: SubscriptionUpdateInput!): Subscription
Update an existing subscription. Takes `SubscriptionUpdateInput`; returns `Subscription` if successful.

Data Types

Subscription
A subscription record.

```
type Subscription {
    entity_id: Int                          Subscription ID
    increment_id: String                    Subscription Increment ID (user-visible)
    quote_id: Int                           Subscription Quote ID
    description: String                     Description
    customer_id: Int                        Customer ID
    created_at: String                      Created-at date
    updated_at: String                      Updated-at date
    store_id: Int                           Store ID
    next_run: String                        Next scheduled run date
    last_run: String                        Last run date
    last_notified: String                   Last upcoming-billing notified date
    subtotal: Float                         Subtotal amount
    length: Int                             Length (total number of installments)
    run_count: Int                          Run count (number of installments completed)
    status: String                          Status
    frequency_count: Int                    Frequency count (how often it runs)
    frequency_unit: String                  Frequency unit (how often it runs)
    additional: [TokenBaseKeyValue]         Subscription metadata
    quote: SubscriptionQuote                Subscription contents/fulfillment details
    logs: [SubscriptionLog]                 Subscription history logs
}
```

SubscriptionQuoteItem
A subscription item.

```
type SubscriptionQuoteItem {
    item_id: Int
    sku: String
    name: String
    description: String
    product_id: Int
    parent_item_id: Int
    is_virtual: Boolean
    weight: Float
    qty: Float
    price: Float
```

```
    custom_price: Float
    discount_amount: Float
    tax_percent: Float
    tax_amount: Float
    row_total: Float
    row_total_with_discount: Float
    product_type: String
    original_custom_price: Float
    price_incl_tax: Float
    row_total_incl_tax: Float
    discount_tax_compensation_amount: Float
    free_shipping: Boolean
    weee_tax_applied_amount: Float
    weee_tax_applied_row_amount: Float
    weee_tax_disposition: Float
    weee_tax_row_disposition: Float
}
```

SubscriptionQuoteBillingAddress

The subscription billing address

```
type SubscriptionQuoteBillingAddress {
    address_id: Int
    customer_address_id: Int
    region: String
    region_id: Int
    country_id: String
    street: [String]
    company: String
    telephone: String
    fax: String
    postcode: String
    city: String
    firstname: String
    lastname: String
    middlename: String
    prefix: String
    suffix: String
    vat_id: String
}
```

SubscriptionQuoteShippingAddress

The subscription shipping address

```
type SubscriptionQuoteShippingAddress {
    address_id: Int
    customer_address_id: Int
    same_as_billing: Boolean
    region: String
    region_id: Int
    country_id: String
    street: [String]
    company: String
    telephone: String
    fax: String
    postcode: String
    city: String
    firstname: String
    lastname: String
    middlename: String
    prefix: String
    suffix: String
    vat_id: String
    shipping_method: String
    shipping_description: String
    weight: Float
    subtotal: Float
    subtotal_with_discount: Float
    tax_amount: Float
    shipping_amount: Float
    shipping_tax_amount: Float
    discount_amount: Float
    grand_total: Float
    customer_notes: String
    discount_description: String
```

```
    shipping_discount_amount: Float
    subtotal_incl_tax: Float
    discount_tax_compensation_amount: Float
    shipping_discount_tax_compensation_amount: Float
    shipping_incl_tax: Float
    free_shipping: Boolean
    customer_balance_amount: Float
}
```

SubscriptionQuotePayment

The subscription payment info

```
type SubscriptionQuotePayment {
    entity_id: Int
    method: String
    cc_type: String
    cc_last_4: String
    cc_owner: String
    cc_exp_month: String
    cc_exp_year: String
    cc_ss_owner: String
    cc_ss_start_month: String
    cc_ss_start_year: String
    po_number: String
    additional_data: [TokenBaseKeyValue]
    cc_ss_issue: String
    additional_information: [TokenBaseKeyValue]
    paypal_payer_id: String
    paypal_payer_status: String
    paypal_correlation_id: String
    tokenbase_id: Int
}
```

SubscriptionLog

A subscription event log

```
type SubscriptionLog {
    log_id: Int
    created_at: String
    status: String
    order_id: Int
    order_increment_id: String
    agent_id: Int
    description: String
    additional_information: [TokenBaseKeyValue]
}
```

SubscriptionPaymentAccount

An available subscription payment option. Note that `public_hash` is used as `payment_account` for the `updateSubscription` mutation (`SubscriptionUpdateInput` object).

```
type SubscriptionPaymentAccount {
    public_hash: String
    label: String
    method: String
    cc_type: String
    cc_last4: String
    cc_exp: String
}
```

TokenBaseKeyValue

Container for generic key/value data.

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

SubscriptionUpdateInput

Input parameters for a subscription update. Allows changing of the payment method (by hash) and shipping address.

Leave out `payment_account` or `shipping_address`/`shipping_address_id` to leave them unchanged.

`shipping_address_id` will override any `shipping_address`, if both are given. Provide one or the other, not both.

```
input SubscriptionUpdateInput {
    entity_id: Int!                         Subscription ID to update (required)
    payment_account: String                 Identifier hash of the TokenBase or Vault record to use for payment.
    shipping_address_id: Int                Customer address ID to change the subscription shipping address to.
    shipping_address: CustomerAddressInput Customer address to change the subscription shipping address to.
}
```

GraphQL Query Examples

Some response data has been omitted for brevity.

Fetch subscription by ID

Example request:

```
{
  subscriptions(entity_id:8) {
    entity_id,
    increment_id,
    quote_id,
    description,
    customer_id,
    created_at,
    updated_at,
    store_id,
    next_run,
    last_run,
    last_notified,
    subtotal,
    length,
    run_count,
    status,
    frequency_count,
    frequency_unit,
    additional {
      key,
      value
    },
    quote {
      entity_id
      customer_email
      subtotal,
      grand_total,
      quote_currency_code,
      items {
        item_id
        sku
        name
        description
        product_id
        parent_item_id
        is_virtual
        weight
        qty
        price
        custom_price
        discount_amount
        tax_percent
        tax_amount
        row_total
        row_total_with_discount
        product_type
        original_custom_price
        price_incl_tax
        row_total_incl_tax
```

```
        discount_tax_compensation_amount
        free_shipping
        weee_tax_applied_amount
        weee_tax_applied_row_amount
        weee_tax_disposition
        weee_tax_row_disposition
      },
      billing_address {
        address_id
        customer_address_id
        region
        region_id
        country_id
        street
        company
        telephone
        fax
        postcode
        city
        firstname
        lastname
        middlename
        prefix
        suffix
        vat_id
      },
      shipping_address {
        address_id
        customer_address_id
        same_as_billing
        region
        region_id
        country_id
        street
        company
        telephone
        fax
        postcode
        city
        firstname
        lastname
        middlename
        prefix
        suffix
        vat_id
        shipping_method
        shipping_description
        weight
        subtotal
        subtotal_with_discount
        tax_amount
        shipping_amount
        shipping_tax_amount
        discount_amount
        grand_total
        customer_notes
        discount_description
        shipping_discount_amount
        subtotal_incl_tax
        discount_tax_compensation_amount
        shipping_discount_tax_compensation_amount
        shipping_incl_tax
        free_shipping
        customer_balance_amount
      },
      payment {
        entity_id
        method
        cc_type
        cc_last_4
        cc_owner
        cc_exp_month
        cc_exp_year
        cc_ss_owner
        cc_ss_start_month
        cc_ss_start_year
        po_number
        additional_data {
          key
          value
```

```
        }
        cc_ss_issue
        additional_information {
          key
          value
        }
        paypal_payer_id
        paypal_payer_status
        paypal_correlation_id
      }
    },
    logs {
      log_id
      created_at
      status
      order_id
      order_increment_id
      agent_id
      description
      additional_information {
        key,
        value
      }
    }
  }
}
```

Example response:

```
{
  "data": {
    "subscriptions": [
      {
        "entity_id": 8,
        "increment_id": "000000008",
        "quote_id": 24,
        "description": "Single-opt Subscription",
        "customer_id": 1,
        "created_at": "2018-12-27 18:21:41",
        "updated_at": "2019-04-24 19:51:16",
        "store_id": 1,
        "next_run": "2020-12-27 18:21:41",
        "last_run": "2018-12-27 18:22:31",
        "last_notified": "0000-00-00 00:00:00",
        "subtotal": 20,
        "length": 5,
        "run_count": 2,
        "status": "active",
        "frequency_count": 1,
        "frequency_unit": "year",
        "additional": [],
        "quote": {
          "entity_id": 24,
          "customer_email": "roni_cost@example.com",
          "subtotal": 20,
          "grand_total": 24.49,
          "quote_currency_code": "USD",
          "items": [
            {
              "item_id": 30,
              "sku": "single-opt",
              "name": "Single-opt Subscription",
              "description": null,
              "product_id": 2048,
              "parent_item_id": null,
              "is_virtual": false,
              "weight": 1,
              "qty": 1,
              "price": 20,
              "custom_price": null,
              "discount_amount": 2,
              "tax_percent": 8.25,
              "tax_amount": 1.49,
              "row_total": 20,
              "row_total_with_discount": 0,
              "product_type": "simple",
              "original_custom_price": null,
              "price_incl_tax": 21.65,
```

```
            "row_total_incl_tax": 21.65,
            "discount_tax_compensation_amount": 0,
            "free_shipping": false,
            "weee_tax_applied_amount": null,
            "weee_tax_applied_row_amount": null,
            "weee_tax_disposition": null,
            "weee_tax_row_disposition": null
          }
        ],
        "billing_address": {
          "address_id": 66,
          "customer_address_id": 1,
          "region": "Michigan",
          "region_id": 33,
          "country_id": "US",
          "street": [
            "6146 Honey Bluff Parkway"
          ],
          "company": null,
          "telephone": "(555) 229-3326",
          "fax": null,
          "postcode": "49628-7978",
          "city": "Calder",
          "firstname": "Veronica",
          "lastname": "Costello",
          "middlename": null,
          "prefix": null,
          "suffix": null,
          "vat_id": null
        },
        "shipping_address": {
          "address_id": 67,
          "customer_address_id": 1,
          "same_as_billing": true,
          "region": "Michigan",
          "region_id": 33,
          "country_id": "US",
          "street": [
            "6146 Honey Bluff Parkway"
          ],
          "company": null,
          "telephone": "(555) 229-3326",
          "fax": null,
          "postcode": "49628-7978",
          "city": "Calder",
          "firstname": "Veronica",
          "lastname": "Costello",
          "middlename": null,
          "prefix": null,
          "suffix": null,
          "vat_id": null,
          "shipping_method": "flatrate_flatrate",
          "shipping_description": "Flat Rate - Fixed",
          "weight": 1,
          "subtotal": 20,
          "subtotal_with_discount": 18,
          "tax_amount": 1.49,
          "shipping_amount": 5,
          "shipping_tax_amount": 0,
          "discount_amount": -2,
          "grand_total": 24.49,
          "customer_notes": null,
          "discount_description": null,
          "shipping_discount_amount": 0,
          "subtotal_incl_tax": 21.65,
          "discount_tax_compensation_amount": 0,
          "shipping_discount_tax_compensation_amount": 0,
          "shipping_incl_tax": 5,
          "free_shipping": false,
          "customer_balance_amount": null
        },
        "payment": {
          "entity_id": null,
          "method": "checkmo",
          "cc_type": null,
          "cc_last_4": null,
          "cc_owner": null,
          "cc_exp_month": null,
          "cc_exp_year": "0",
          "cc_ss_owner": null,
```

```
            "cc_ss_start_month": "0",
            "cc_ss_start_year": "0",
            "po_number": null,
            "additional_data": [],
            "cc_ss_issue": null,
            "additional_information": [
              {
                "key": "method_title",
                "value": "Check / Money order"
              },
              {
                "key": "is_subscription_generated",
                "value": "1"
              }
            ],
            "paypal_payer_id": null,
            "paypal_payer_status": null,
            "paypal_correlation_id": null
          }
        },
        "logs": [
          {
            "log_id": 13,
            "created_at": "2018-12-27 18:21:41",
            "status": "active",
            "order_id": 19,
            "order_increment_id": "000000019",
            "agent_id": 0,
            "description": "Subscription created. Initial order total: $29.36",
            "additional_information": []
          },
          {
            "log_id": 14,
            "created_at": "2018-12-27 18:22:31",
            "status": "active",
            "order_id": 20,
            "order_increment_id": "000000020",
            "agent_id": 1,
            "description": "Subscription billed. Order total: $24.49",
            "additional_information": []
          }
        ]
      }
    ]
  }
}
```


Fetch payment options

Example request:

```
{
  subscriptionPayments(subscription_id:110) {
    label
    method
    public_hash
    cc_exp
    cc_type
    cc_last4
  }
}
```

Example response:


```
{
  "data": {
    "subscriptionPayments": [
      {
        "label": "Discover XXXX-0012",
        "method": "authnetcim",
        "public_hash": "5ef24e8525df42f2b3e0967566cd236aecc358e3",
        "cc_exp": "2022-12-31 23:59:59",
        "cc_type": "DI",
        "cc_last4": "0012"
      },
      {
        "label": "Visa XXXX-0027",
```

```
        "method": "paradoxlabs_cybersource",
        "public_hash": "ad2058eabe2e8c78470b14cf53cffb0ec2b84c4d",
        "cc_exp": "2022-12-31 23:59:59",
        "cc_type": "VI",
        "cc_last4": "0027"
      },
      {
        "label": "VI XXXX-1111",
        "method": "braintree",
        "public_hash": "8cf50d80c55256758eecaab579d24628c73a7edc8dc786f9330f9a794e2cb43b",
        "cc_exp": "2023-01-31T23:59:59+00:00",
        "cc_type": "VI",
        "cc_last4": "1111"
      },
      {
        "label": "Purchase Order",
        "method": "purchaseorder",
        "public_hash": "purchaseorder",
        "cc_exp": "",
        "cc_type": null,
        "cc_last4": null
      }
    ]
  }
}
```


Change subscription status

Example request:

```
mutation {
  changeSubscriptionStatus(entity_id:8, status:"paused") {
    entity_id,
    increment_id,
    quote_id,
    description,
    customer_id,
    created_at,
    updated_at,
    store_id,
    next_run,
    last_run,
    last_notified,
    subtotal,
    length,
    run_count,
    status,
    frequency_count,
    frequency_unit
  }
}
```

Example response:


```
{
  "data": {
    "changeSubscriptionStatus": {
      "entity_id": 8,
      "increment_id": "000000008",
      "quote_id": 24,
      "description": "Single-opt Subscription",
      "customer_id": 1,
      "created_at": "2018-12-27 18:21:41",
      "updated_at": "2019-04-25 04:00:02",
      "store_id": 1,
      "next_run": "2020-12-27 18:21:41",
      "last_run": "2018-12-27 18:22:31",
      "last_notified": "0000-00-00 00:00:00",
      "subtotal": 20,
      "length": 5,
      "run_count": 2,
      "status": "paused",
      "frequency_count": 1,
      "frequency_unit": "year"
    }
  }
}
```

Update subscription

Example request:

```
mutation {
  updateSubscription(input: {
    entity_id:8,
    payment_account:"88bb7dc06faad55c77177446ed83047811234008",
    shipping_address: {
        region: {
          region_code: "PA",
          region: "Pennsylvania",
          region_id: 51
        },
        country_id: US,
        street: [
          "123 Test Lane"
        ],
        telephone: "111-111-1111",
        postcode: "12345",
        city: "Lancaster",
        firstname: "John",
        lastname: "Doe"
      }
    }
  ) {
    entity_id,
    increment_id,
    description,
    status,
    quote {
      shipping_address {
        customer_address_id
        region
        country_id
        street
        telephone
        fax
        postcode
        city
        firstname
        lastname
      },
      payment {
        method
        cc_type
        cc_last_4
        cc_exp_month
        cc_exp_year
      }
    }
  }
}
```

Example response:

```
{
  "data": {
    "updateSubscription": {
      "entity_id": 8,
      "increment_id": "000000008",
      "quote_id": 24,
      "description": "Single-opt Subscription",
      "status": "paused",
      "quote": {
        "shipping_address": {
          "customer_address_id": null,
          "region": "Pennsylvania",
          "country_id": "US",
          "street": [
            "123 Test Lane"
          ],
          "company": "",
          "telephone": "111-111-1111",
          "fax": null,
          "postcode": "12345",
          "city": "Lancaster",
          "firstname": "John",
```

```
          "lastname": "Doe"
        },
        "payment": {
          "method": "authnetcim",
          "cc_type": "DI",
          "cc_last_4": "0027",
          "cc_exp_month": "03",
          "cc_exp_year": "2021"
        }
      }
    }
  }
}
```

Product Setup by PHP

One way to manage the setup and configuration of a large number of subscription products is with PHP. Here is some sample code, expected to be run in the context of Magento 2 with a loaded product and product repository. Note that this code is **for demonstration only**, and must be modified to fit your specific use case and requirements.

```php
<?php
// Example code for demonstration only -- must be run in M2 context, with ProductRepository
$productId = 123;
$intervals = [
    [
        'frequency_count' => '30',
        'frequency_unit' => 'day',
        'length' => null,
        'installment_price' => 15.00,
        'adjustment_price' => null,
    ],
    [
        'frequency_count' => '60',
        'frequency_unit' => 'day',
        'length' => null,
        'installment_price' => 13.50,
        'adjustment_price' => null,
    ],
];

$product = $this->productRepository->getById($productId);
$product->setData('subscription_intervals_grid', $intervals);
$product->setData('subscription_active', 1);
$product->setData('subscription_allow_onetime', 1);
$this->productRepository->save($product);
```

This example loads product with ID 123, enables subscription billing with one-time purchasing allowed, and adds two subscription options (30 days at $15, 60 days at $13.50). This is exactly equivalent to entering the same data into the admin product edit form in the “Subscription Options” section. At save, the interval options data will be saved, the associated custom option will be generated or updated, and Magento should automatically update the associated product page after indexing the product changes.

This method leverages the same internal code used to process the admin product edit form. The `subscription_intervals_grid` values can be passed in as an array or the JSON-encoded equivalent.

Split Database

This module fully supports Magento Commerce's split database feature, but note that this feature is deprecated as of Magento 2.4 and likely to be removed in the near future. We strongly recommend setting up the split database environment prior to installing this module, but no special setup should be needed otherwise. The `paradoxlabs_subscription` and `paradoxlabs_subscription_log` tables will be added to the `checkout` database for proximity to the `quote` tables. `paradoxlabs_subscription_product_interval` will be added to the primary (catalog) database.

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