Getting Started

Cashier API Documentation

Contents

Introduction

A plugin is a software component that will allow you to add a specific feature to your checkout. For example, you can add, remove or modify data in the checkout, override existing functionality, add new functionality, and communicate with external apps; all inside of the checkout process.

In this guide we will take you through each step of how to build your own plugin for Cashier. You can use our pre-built "*node-example-plugin*" as a reference to follow along. This example plugin utilizes the 8.x version of NodeJS and the 4.x version of Express JS. We will incrementally add functionality to this plugin following recommended best practices.

Let's get started!

Prerequisites

Step 1 - Install Cashier

Install Bold Cashier on your development store.

- Download and Install Cashier

Step 2 - Contact us

Email partners@boldcommerce.com with any questions you have about your plugin idea and to make sure that what you want to achieve is feasible.

Feasible:

- A plugin that applies gift cards to an order.
- A plugin that provides custom shipping rates.
- A plugin that overrides the taxes of an order.
- A plugin that adds, removes, or edits items in the cart.

Not Feasible:

- Applying JavaScript to any non-order data during checkout

Consideration: If you require the use of our external API in tandem with a plugin, you will need an external platform application to handle calling those endpoints.

You will be sent a plugin request form that you will need to completely fill out in order to be a Cashier integrator.

Step 3 - Request to be a Cashier integrator

Fill out and submit the plugin request form and get approved to build your plugin.
Fully completed applications take up to 1 business day to process. Once your application has been processed we will provide you with your client/plugin UUID and secret authentication information. We will then make your plugin privately available to you on your development store through Cashier's Marketplace.

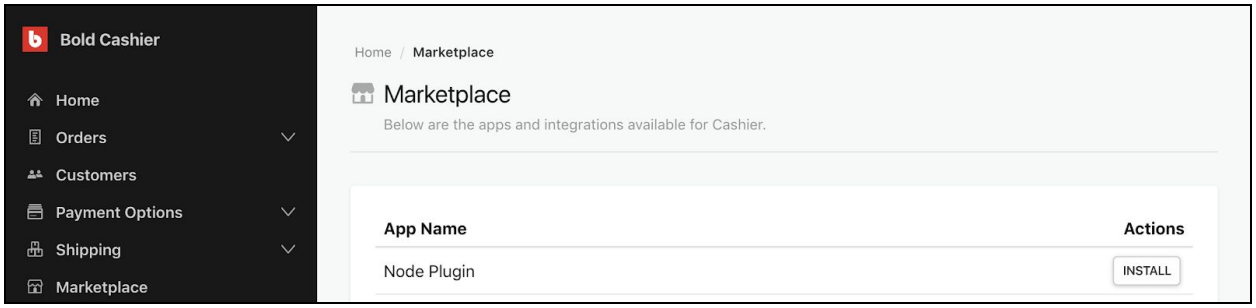
If you haven't already installed Cashier, please download and install Cashier now.

The client/plugin UUID and secret values will not change as they have a one-to-one relationship to the plugin, so they should remain as environment variables in your plugin even once the plugin is in production.

Plugin Installation Setup

Setup the Install Redirect URL

Your plugin should now be available in the Marketplace inside the Cashier App admin with an install button next to it. Clicking the install button will cause Cashier to redirect the shop owner's browser to the "Install Redirect URL" that you submitted in the plugin request form.



- Example of an Install Redirect endpoint on GitHub

NOTE: Please see the array of scopes in the endpoint linked above. These are the scopes you're requesting from Cashier and they govern what endpoints and actions your plugin is allowed to use. Every documented endpoint and action has it's required scope listed below it. Scopes are only applied on install so if, after installing, you want to add more scopes, you need to reinstall your plugin for them to take effect.

Cashier adds a query string to the request:

- `platform:` the eCommerce platform which the shop belongs to
- `shop:` the platform-unique identifier for the shop

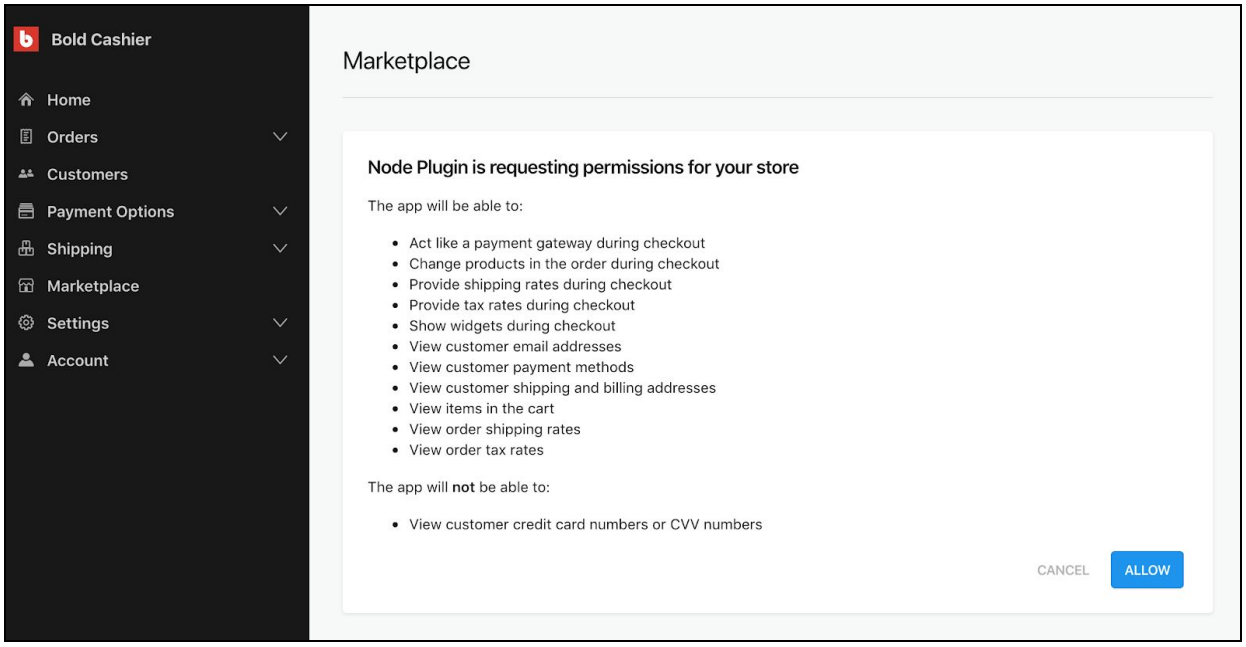
This endpoint is responsible for redirecting you, the shop owner, back to the Cashier App with the plugin's desired API scopes:

- POST request: `https://cashier.boldcommerce.com/api/v1/{platform}/{shop}/oauth/authorize?client_id={client_id}&scope={scope}&response_type=code`

The parameters are as follows:

- Where `{platform}` and `{shop}` are replaced by their corresponding values from the initial request
- Where `{client_id}` is replaced by the plugin's unique client identifier issued by Cashier
- Where `{scope}` is replaced by a space-delimited list of desired API scopes

If the "Install Redirect URL" returns the correct response Cashier will then redirect you to a page where you accept all the scopes (pictured below) you defined in the scopes array described above.



Setup the Authorize Redirect URL

Clicking "Allow" on that page will cause Cashier to hit the "Authorize Redirect URL" you submitted in the plugin request form. If the shop owner accepts the scope requested by the plugin, they will be redirected to this endpoint.

- Example node of Authorize Redirect endpoint on GitHub
- Example of a request & response body

Example Request Body

```
{
    "client_id": "d5cb40ab-05af-4168-b1e0-a37660824779",
    "client_secret":
"kjjcEyJdRzn1dusPU5aAChHZC1s3DWHS0ZdRKgxxIUbEPf6wqPRZTpJjUPmUj116",
    "code":
"mnU8xUaGr1ACMDl793tnJEN7gBmT9SN8YLFcmRh7q72hIsy61HeYfHRUyjW0SxSL",
    "grant_type": "authorization_code"
}
```

Example Response Body

```
{
    "shop": "agitated-darwin.myshopify.com",
    "platform": "shopify",
    "access_token":
"yPdckrPdqOHkyBkx0spIgNhXLi5jI8uXaNxXvJqOJ7g0N2ljAvJqBGpgEsclsfkt",
    "scope": "add_tags provide_discounts",
    "token_type": "bearer"
}
```

NOTE: in this endpoint you can choose where to redirect your users. `res.redirect(`https://${domain}/admin/${platform}/${shop}/marketplace`);` redirects your users back to the marketplace page.

Cashier adds a query string to the request:

- platform: the eCommerce platform which the shop belongs to
- shop: the platform-unique identifier for the shop
- code: a temporary authorization code

The plugin can call Cashier's API to exchange the authorization code for an access token:

- POST https://cashier.boldcommerce.com/api/v1/{platform}/{shop}/oauth/access_token

with the following request body:

- client_id: the plugin's unique client identifier issued by Cashier
- client_secret: the plugin's secret key issued by Cashier
- code: the provided authorization code
- grant_type: the exact string “authorization_code”

If the authorization code is valid, Cashier will respond with the following key-value pairs:

- shop: the platform-unique identifier for the shop
- platform: the eCommerce platform which the shop belongs to
- access_token: plugin’s unique access token for Bold’s backend API. Note: this will be regenerated on every install.
- scope: list of permissions that your plugin requires
- token_type:

If the "Authorize Redirect URL" returns the correct response Cashier completes the installation.

Events & Actions

Events

Cashier sends out events at different times during the checkout process to any plugins that are registered to listen for them. During the “Request to be a Cashier integrator” step, you specified which events you wanted your plugin to listen for using the plugin request form. A plugin will not receive an event it isn't registered to listen for. If you would like to register your plugin to listen for additional events, please contact us.

- A list of all the events, when they trigger, and how they work

Plugins can manipulate a customer's ongoing checkout session by sending actions in response to events.

Actions

Actions are how a plugin interacts with the checkout process in Cashier. When a plugin receives an event from Cashier it can respond with an action. Cashier will receive that action and update the checkout as specified by that action.

- A list of actions, the scopes they require, and how to use them

Example

Here is an example of what responding to an event with an action could look like:

Example Event Dispatch Endpoint URL

All events that your plugin is registered to receive will be sent to your plugin's "Event Dispatch Endpoint URL" that you submitted in the plugin request form.

- Example

Example Event Handler Function

The endpoint passes every event to an event handler function like this where each event is handled separately

- Example

Example Initialize Checkout

For this example, let's look at the `handleInitializeCheckout` function for a common usage of the `initialize_checkout` event.

- Example

Example Action

`initialize_checkout` occurs once when a customer enters checkout and in this example we are responding to that with the `APP_UPDATE_WIDGET` action. This action creates a widget somewhere in the checkout (based on the `position` parameter).

Widgets will be explained in more detail in the next section of the guide, for now just take note that checkout is being modified with these actions.

Note how we are returning three of the same types of actions in response to the `initialize_checkout` event. You can return as many actions as you want in response to

any event, including multiple of the same type of action. Once you return an action it will be executed immediately.

Adding widgets to checkout is only one example of what you can do with actions. Refer back to the documentation linked above to get an idea of what else can be done with events and actions.

Widgets

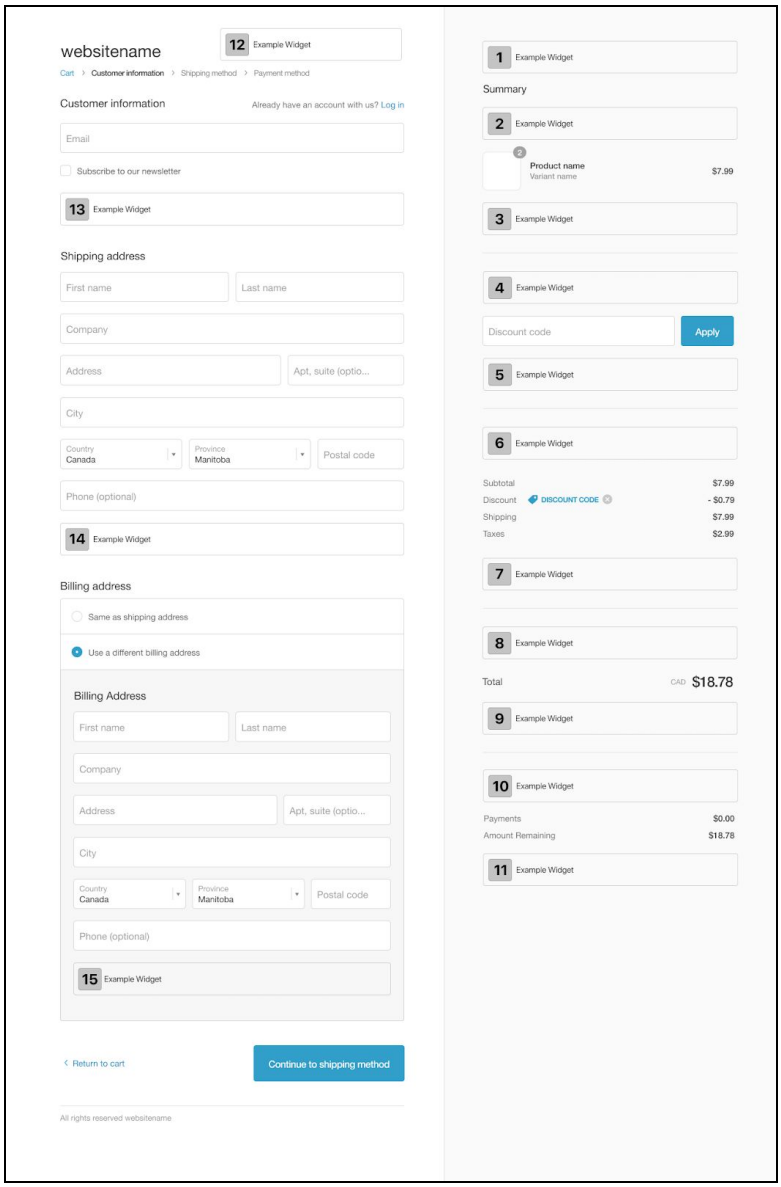
In the previous section we showed an example of executing the 'APP_UPDATE_WIDGET' action. This is the action used to create and modify widgets on the checkout page.

Widget Positions

The value of the position parameter determines where the widget will show up in the checkout.

- More documentation on position parameters

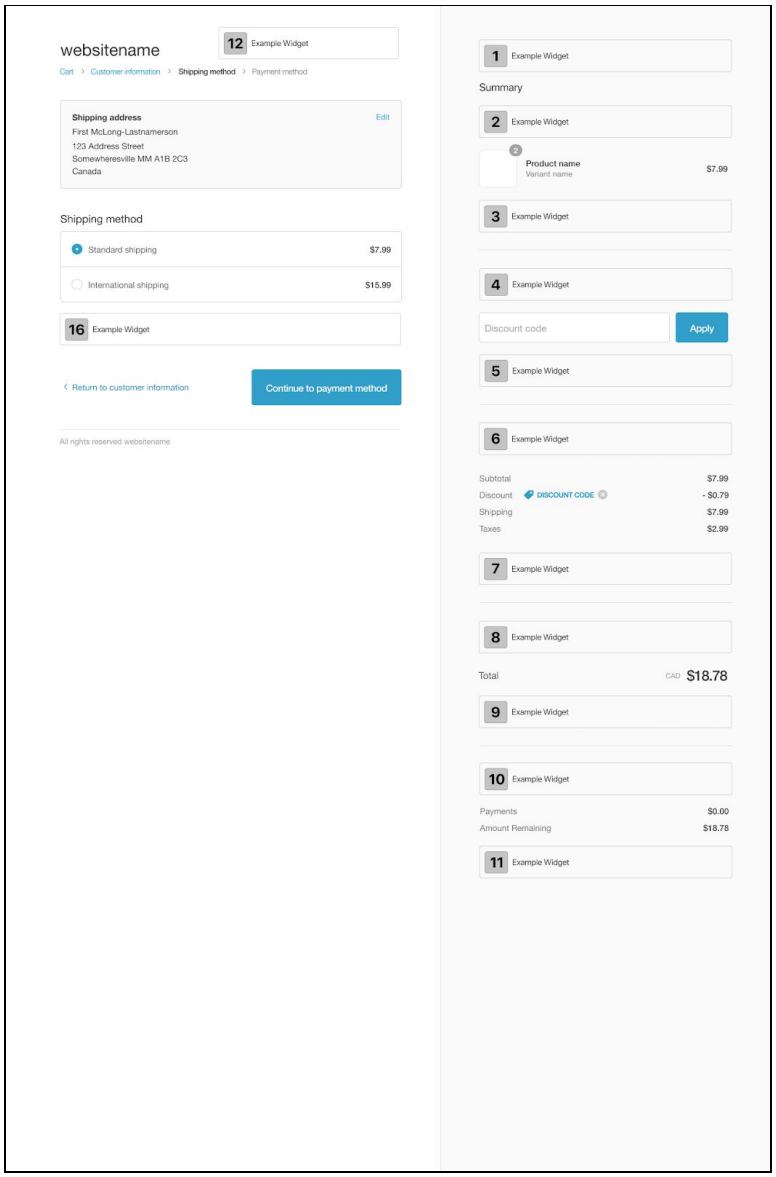
Three Page Checkout 1/3



Legend

1. summary_above_header
2. summary_top
3. line_items
4. discount
5. discount_top
6. price_summary_top
7. price_summary
8. total_top
9. total
10. payments_top
11. payments
12. header
13. customer_info
14. shipping
15. billing

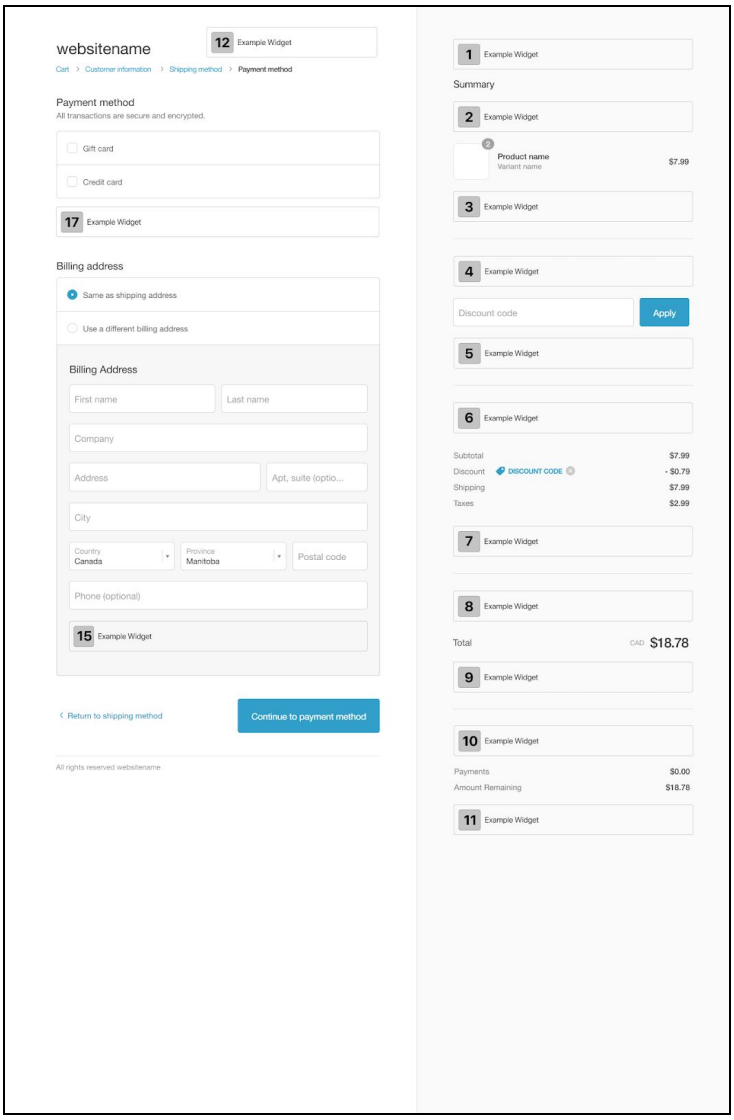
Three Page Checkout 2/3



Legend

1. summary_above_header
2. summary_top
3. line_items
4. discount
5. discount_top
6. price_summary_top
7. price_summary
8. total_top
9. total
10. payments_top
11. payments
12. header
13. customer_info
14. shipping
15. billing
16. shipping_lines

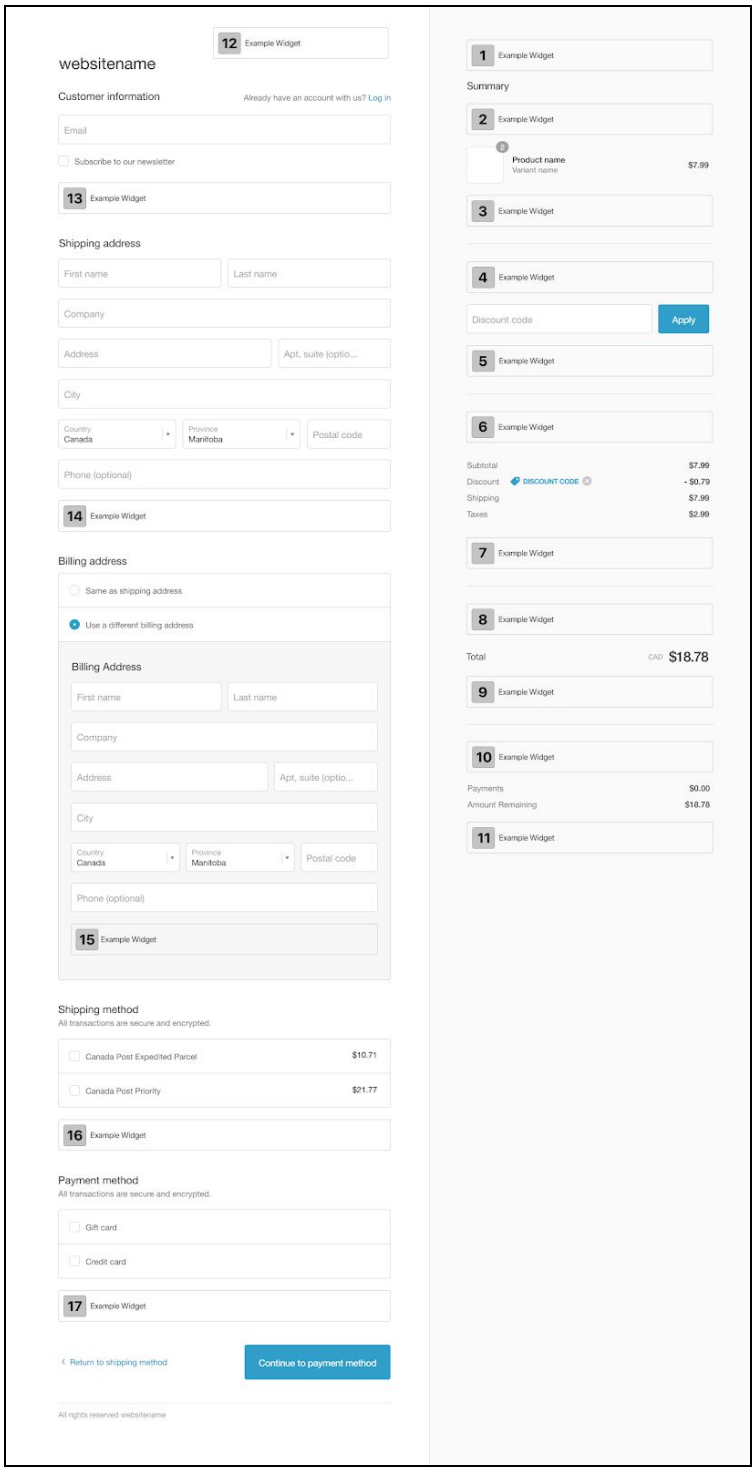
Three Page Checkout 3/3



Legend

1. summary_above_header
2. summary_top
3. line_items
4. discount
5. discount_top
6. price_summary_top
7. price_summary
8. total_top
9. total
10. payments_top
11. payments
12. header
13. customer_info
14. shipping
15. billing
16. shipping_lines
17. payment_gateway

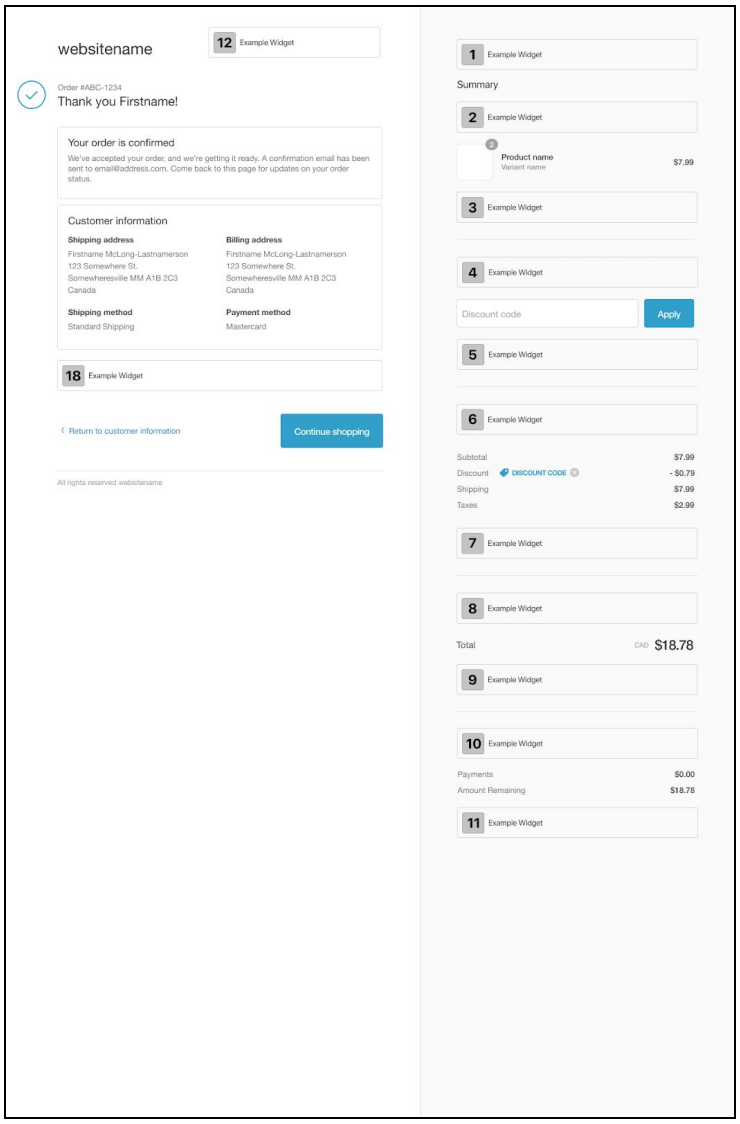
One Page Checkout



Legend

1. summary_above_header
2. summary_top
3. line_items
4. discount
5. discount_top
6. price_summary_top
7. price_summary
8. total_top
9. total
10. payments_top
11. payments
12. header
13. customer_info
14. shipping
15. billing
16. shipping_lines
17. payment_gateway

Thank You Page Checkout



Legend

1. summary_above_header
2. summary_top
3. line_items
4. discount
5. discount_top
6. price_summary_top
7. price_summary
8. total_top
9. total
10. payments_top
11. payments
12. header
13. customer_info
14. shipping
15. billing
16. shipping_lines
17. payment_gateway
18. thank_you

Widget Types

There are 4 types of widget you can use 'app_hook', 'external_link', 'iframe', 'modal'.

- Examples & more documentation on widget types

App Hook Widget

'app_hook' widgets are simple widgets that can display text along with an icon. When you click them they trigger an 'app_hook' event in your plugin. That event also comes with a specific 'click_hook' parameter that lets you determine exactly which 'app_hook' widget that the 'app_hook' event is coming from.

Example APP_UPDATE_WIDGET action

```
{
    "type": "APP_UPDATE_WIDGET",
    "data": {
        "name": "example_widget",
        "type": "external_link",
        "position": "line_items",
        "text": "Click here to read our return policy",
        "icon": "https://example.myshopify.com/icon.png",
        "link_url": "https://example.myshopify.com/returns"
    }
}
```

External Link Widget

'external_link' widgets are another simple kind of widget that, like the 'app_hook' widget, can display text and an icon. When you click on this widget it will open a new window/tab to the link specified in the 'link_url' parameter.

Iframe Widget

'Iframe' widgets are more versatile since they allow you to embed an external page into the widget. The 'source' parameter for this widget will be the url to the external page you want to display and the 'frame_origin' will be the url to your plugin. More details about what you can do with the embedded pages will be discussed later in the guide.

Modal Widget

'modal' widgets are very similar to the 'iframe' widgets. The only difference is that they look like 'app_hook' widgets but when you click on them they open an embedded page within a dialog box. The page is specified though the 'click_url' parameter.

NOTE: Both 'iframe' and 'modal' widgets have a way to communicate with Cashier through their embedded pages. With these messages 'iframe' and 'modal' widgets can change size, send 'app_hook's and request the current state of the order data in checkout.

- Messages that widgets can send & receive

```
parent.postMessage({
    type: 'checkout/app_hook',
    payload: {
        hook: 'example_hook',
        data: { foo: 'bar' }
    }
}, '*');

parent.postMessage({
    type: 'checkout/resize_frame',
    payload: { height: 100 }
}, '*');

parent.postMessage({
    type: 'checkout/close_frame'
}, '*');

parent.postMessage({
    type: 'checkout/request_order'
}, '*');
window.addEventListener('message', function (event) {
    let message = event.data;
    switch (message.type) {
        case 'checkout/receive_order':
            console.log('Received order data', message.payload);
            break;
    }
});
```

Example Widget

Here is an example implementation of a widget (see below for an example)

The widget starts listening for message events in order to receive the `checkout/receive_order` message later.

It sends `checkout/resize_frame` to resize its container to 100px in height. This makes the widget visible in the checkout. Widgets are hidden until they resize themselves for the first time.

It sends `checkout/request_order` to request the current order data from Cashier.

Cashier responds to the widget by sending `checkout/receive_order` with the current order data.

The `message` event listener is called, which logs the order data to the browser console.

Example Embedded Page:

```
<!doctype html>
<html>

<head>
<meta charset="utf-8">
<script>
    // Start listening for messages from Cashier:
    window.addEventListener('message', function (event) {
        let message = event.data;
        switch (message.type) {
            case 'checkout/receive_order':
                console.log('Received order data', message.payload);
                break;
        }
    });

    // Set the height of the iframe containing this embedded page:
    parent.postMessage({
        type: 'checkout/resize_frame',
        payload: { height: 100 }
    }, '*');

    // Request order data from Cashier:
    parent.postMessage({
        type: 'checkout/request_order'
    }, '*');
</script>
</head>

<body>
<h1>Hello, Cashier!</h1>
</body>

</html>
```

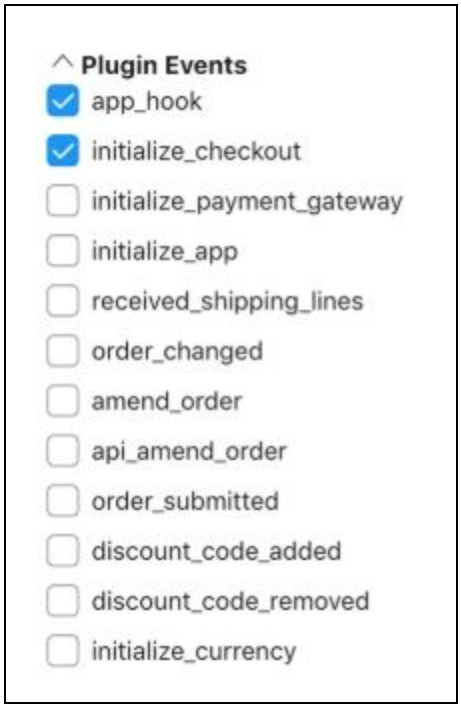
Complete an Order

How to use events and widgets to apply a discount to an order.

This guide will give you an overview on how to send a basic app_hook event. `app_hook` events are very useful for Cashier plugins as they allow you to define your own behaviour within the checkout flow.

Listen for Events

Make sure your plugin listens for the 'initialize_checkout' and 'app_hook' events.



Respond to Events

Respond to the `'initialize_checkout'` event with an `'APP_UPDATE_WIDGET'` action. The widget `'type'` should be `'app_hook'`, all the other information is up to you.

```
function handleEvent(req) {
  switch (req.body.event) {
    case 'initialize_checkout':
      return handleInitializeCheckout(req);
    case 'app_hook':
      return handleAppHook(req);
    default:
      return [];
  }
};

function handleInitializeCheckout(req) {
  return [
    {
      type: 'APP_UPDATE_WIDGET',
      data: {
        name: 'my_discount_widget',
        type: 'app_hook',
        position: 'discount',
        text: 'Discount 5%',
        icon: 'https://via.placeholder.com/50x50.png',
        click_hook: 'apply_discount',
      },
    },
  ];
}
```

The above `APP_UPDATE_WIDGET` action creates a widget near the discount form when checkout initializes. When you click it, it sends an `app_hook` with the type `apply_discount`, as specified in the `click_hook` parameter.

Next we're going to respond to the 'app_hook' event by listening for the specific 'click_hook' you set up in the previous step. In this example we're listening for the 'apply_discount' hook.

We'll respond to that click hook with two separate actions. The 'DISCOUNT_CART' and 'APP_UPDATE_WIDGET' actions. The discount action will apply a discount directly to the cart and the update widget action will change the text on the widget to let the customer know they already clicked it.

```
function handleAppHook(req) {
  switch (req.body.properties.hook) {
    case 'apply_discount':
      return [
        {
          type: 'DISCOUNT_CART',
          data: {
            discountPercentage: 5,
            transformationMessage: 'Money saved',
          },
        },
        {
          type: 'APP_UPDATE_WIDGET',
          data: {
            name: 'my_discount_widget',
            type: 'app_hook',
            position: 'discount',
            text: 'You\'re welcome',
            click_hook: 'already_used',
            icon: 'https://via.placeholder.com/50x50.png',
          },
        },
      ];
    default:
      return [];
  }
}
```

The `DISCOUNT_CART` action sent back will add a 5% discount to the cart.

The `APP_UPDATE_WIDGET` action will essentially change how your widget looks, since only one widget can exist in the discount position at a time. It will also send a new `app_hook` with the type `already_used` as specified in the `click_hook` parameter. You can now respond to this new `app_hook` in any way you choose.

Try it Yourself

Enter your checkout and click on your 'app_hook' widget. It should apply the discount and change the look of the widget. Complete your order and you'll see that the discount is applied like any other discount code.

Frequently Asked Questions

General

1. **Do I need to build a plugin?**

   You will need to build a plugin if what you want to do requires that you listen to events during the checkout process and/or if you need a widget to appear in the checkout process.

2. **I have a plugin idea, will Bold subsidize the cost of development?**

   For subsidization opportunities, please reach out to partners@boldcommerce.com.

3. **Can I charge for my plugin?**

   Please reach out to partners@boldcommerce.com for more information about charging for your plugin.

4. **Can I make my plugin private/public?**

   Yes, you can choose to only enable it for certain stores or make it available to everyone who installs Cashier.

5. **Can I maintain a duplicate version of my plugin for testing purposes?**

   Yes, you can have multiple versions of your plugin.

6. **What UI guidelines does my widget have to follow?**

   It should be responsive and readable on all devices, including mobile.

7. **What is the process for submitting my completed plugin (QA, etc)?**

   Please contact us at partners@boldcommerce.com when your plugin is complete or when you are nearing your own QA process and we will coordinate from there.

8. **Now that I have a plugin, can I work with Bold to market it?**

   Please contact us at partners@boldcommerce.com to discuss marketing possibilities.

9. **Are discounts applied before or after shipping and taxes?**

   Discounts are only applied to the subtotal of the order.

10. **Will Bold be doing any QA/Testing on my plugin before going live?**

    Bold will be involved in validating that the plugin is ready for use with Cashier (security, performance, etc). You will be responsible for doing a full QA of the functionality of the app, including performance and security, and how it may work with other integrations.

    We may do a quick run through of the checkout process, to ensure that the plugin doesn't crash or doesn't do what you've described, which would prevent us from making the plugin live.

    If the plugin is intended to be available publicly, we may bring in one of our QA specialists to support you, but it will still be up to your to ensure the plugin has been fully tested.

Troubleshooting

1. **I'm getting an error from hitting an endpoint with a 403 response code**

   You are most likely missing a required scope that applies to the endpoint. Please refer to the documentation and make sure you have the scope listed below the endpoint you want to use.

2. **I'm not getting any Cashier events during checkout**

   There are a few things that could be causing that issue, they are listed as follows: Make sure you've subscribed to the events you want and submitted a proper event dispatch URL in the Plugin Request Form.

Make sure your plugin has been successfully installed.

Make sure your website's certificate is up to date.

Make sure your server's ModSecurity rules are properly configured.

3. **I'm running into issues, who can help?**

   Please contact us at partners@boldcommerce.com and we will be able to put you into contact with our assisting team.