Custom Checkout Grouping

If you have Ship to Multiple Addresses in General Settings enabled, you can use the embedded.commerce.checkouts.groupings.after action in the commerce.checkouts domain to customize how your system groups items during checkout and provide unique shipping instructions for certain kinds of items.

Use Case: Create Sub-Groups Based on Special Shipping Needs

By default, all items going to the same destination use the same shipping method, but that group of items may contain a type of item that requires a different shipping method.

For example, a shopper might order a hammer, screwdriver, and ladder to ship to the same destination, but the ladder will require freight shipping since it is an oversized item. If the ladder stays in the same group, the system limits that entire group to only those shipping options that support freight.

Instead, you can customize your system's grouping logic to look for items requiring special shipping methods and put them into their own group so that they do not limit the shipping methods available for the other items in the order.

For more detailed information on default item grouping, see the Grouping section of the Multiple Shipments API Overview Guide.

Usage Examples

Split Groupings Based on Product Property

Using the following code sample you can use the groupings.after action to split groupings based on a product property. For example, if a product has the property “oversized”, a new group is created for that product.

```
var _ = require('underscore');

module.exports = function (context, callback) {
console.log('Starting.');
var orderItems = context.get.orderItems();
var groupings = context.get.checkoutGroupings();
console.log("Number of items in checkout: ", orderItems.length);
console.log("Number of groupings in checkout: ", groupings.length);

if (orderItems.length > 1) {
orderItems.forEach(function (item) {
// Only want to mess with groups that have an oversized item.
if (!isProductOversized(item.product)) return;

console.log("Found oversized item. Checking if groups are valid.");

// Find the group the item belongs to
var existingGroup = _.find(groupings, function (group) {
return _.contains(group.orderItemIds, item.id);
});

// If a buy-one-get-one discount is in effect, a product may be split into 2 line items. We're ignorin
```

```
// If a buy-one-get-one discount is in effect, a product may be split into 2 line items. We're ignorin
g that scenario in this sample.
// If the group only has a single item, no need to break it up further.
if (existingGroup && existingGroup.orderItemIds.length > 1) {
console.log("Splitting group with items: ", existingGroup.orderItemIds);
// Create a new group
var newGroup = createNewGroup(item);
console.log("New group created for item ", item.id);
groupings.push(newGroup);

console.log("Removing the item from existing group.");
var filterIds = _.reject(existingGroup.orderItemIds, function (itemId) { return itemId === item.id;
});
existingGroup.orderItemIds = filterIds;
console.log("Group items after removing oversized item: ", existingGroup.orderItemIds);
}
});
}

// Pickup and digital groups normally get combined with one of the ship groups to create an order.
// You can override this behavior by setting the standaloneGroup flag on a pickup/digital group.

console.log("Total number of groupings: ", groupings.length);
// Remember to set the new groupings!
context.exec.setGroupings(groupings);
console.log("Finished.");
callback();
};

function isProductOversized(product) {
var isOverSizedItem = _.find(product.properties, function (property) {
if (property.attributeFQN === "tenant~isoversized") {
return _.findWhere(property.values, { "Value": true });
}
return false;
});
return isOverSizedItem;
}

function createNewGroup(item) {
var newGroup = {
orderItemIds: [item.id],
destinationId: item.destinationId,
fulfillmentMethod: item.fulfillmentMethod
};
return newGroup;
}
```

Override Default Pickup Groupings

By default, all pickup groups are combined with one of the ship groups to create an order.

In this example, a groupings.after action separates pickup items into groups per pickup location and sets the standaloneGroup flag on those groups to ensure that they’re not combined with ship items when creating orders.

```
var _ = require('underscore');

module.exports = function (context, callback) {
```

```
console.log('Starting.');
var orderItems = context.get.orderItems();
var groupings = context.get.checkoutGroupings();
console.log("Number of items in checkout: ", orderItems.length);
console.log("Number of groupings in checkout: ", groupings.length);

// Get just the pickup groups.
// Note that if there's a digital group, it will be combined with one of the ship groups when creating
orders.
// If you want the digital group to not be combined, set the standaloneGroup flag on that group.
var pickupGroups = _.filter(groupings, function (item) {
return item.fulfillmentMethod === "Pickup";
});

pickupGroups.forEach(function (group) {
// Specifies that this group should not be combined with any others when creating orders.
group.standaloneGroup = true;

// If the group only has a single item, no need to break it up further.
if (group.orderItemIds.length > 1) {

// Check if orderItemIds have different fulfillment location
var groupItems = _.filter(orderItems, function (item) {
return _.contains(group.orderItemIds, item.id);
});

var itemsByFulfillment = _.groupBy(groupItems, 'fulfillmentLocationCode');
// We want to have a group for each fulfillment location.
// Ignore the first set since that will be whatever hasn't been pulled from the current group.
var remainingItemSets = _.toArray(itemsByFulfillment).slice(1);

_.each(remainingItemSets,
function (items) {
console.log("Creating new pick up group for ", items[0].fulfillmentLocationCode);
var newGroup = {
orderItemIds: [],
destinationId: group.destinationId,
fulfillmentMethod: group.fulfillmentMethod,
standaloneGroup: true
};
items.forEach(function (item) {
newGroup.orderItemIds.push(item.id);
console.log("Removing item ", item.id, " from existing group");
var filterIds = _.reject(group.orderItemIds, function (itemId) { return itemId === item.id; });
group.orderItemIds = filterIds;
});
console.log("New group items: ", newGroup.orderItemIds);
groupings.push(newGroup);
}
);
}
});

console.log("Total number of groupings: ", groupings.length);
// Remember to set the new groupings!
context.exec.setGroupings(groupings);
console.log("Finished.");
callback();
};
```