

Gift Certificates

Gift certificates can be such a hassle that you should consider yourself lucky if you can omit them entirely. However, it will be more likely that this will be a “must have” feature, regardless of the time and effort required to implement it. It’s not that gift certificates are inherently that complex, or that the logic surrounding them is impenetrable. Rather, the problem with gift certificates is the number of accommodations that must be made for them throughout your application.

Gift certificates are simply another method of payment, though one for which you are responsible for keeping track of the available balance and for supplying all the functionality for doing debits and credits. Be aware that gift certificates need to be treated just as carefully as you would treat records of money, since gift certificates **are** money. This means that you need to know when they are created, when they are used, etc. In short, **you** become the bank responsible for keeping track of the money the gift certificates represent. This affects not only your database design, but it also affects your accounting reporting, as funds received for gift certificates that have not yet been used for purchasing goods must be tracked as an outstanding liability. Since the laws vary from state on state on whether and when you can “expire” those unredeemed gift certificates, removing the liability and recognizing the revenue, a gift certificate you sell and create a record for today can be around for a long, long time.

The first area in which gift certificates require special consideration is that they are likely the only type of product in your database that aren’t taxable. So, you need to add a validation separate from the standard validations for Products that enforces this condition:

```
class GiftCertificate < Product
  def validate
    if self.taxable?
      errors.add(:taxable, 'can not be set for gift certificates')
    end
  end
end
```

Additionally, you need to check that **taxable?** boolean again when you are calculating the tax as a line item is being added to you cart. See the section on taxes for an example of how the tax calculation code needs to check this field.

Another area in which gift certificates make your life more interesting is dealing with returns, refunds, and when adjusting the quantities of a sale after the sale has been created by the customer. If there are gift certificates associated with a sale, you’ll need to make sure to credit them to increase the amount available to the customer for future purchases. This has to happen even when the credit card would not be refunded at the same time due to the item not having been shipped yet (quantity reduction, cancellation before fulfillment) and thus no capture having been completed.

Creating Gift Certificates

The `Voucher` model represents a `GiftCertificate` purchase, and it has methods like `sale_balance` to track how much of the original balance is left to be spent, and `token`, which is a GUID generated at the time of creation and used to redeem the `Voucher`. In our store, we wait until a shipment is approved before creating vouchers for the customer, even if the “shipment” is purely a digital one (when the customer’s order contains only gift certificates). Alternatively, you could create vouchers immediately after the sale is created.

```
def Shipment
  def create_vouchers
    return unless (certificates = self.line_items.select {|item|
      item.product.is_a?(GiftCertificate)}).any?
    certificates.each do |certificate|
      1.upto(certificate.quantity) do
        voucher = Voucher.create(
          :purchase_amount => certificate.price,
          :sale_balance => certificate.price,
          :shipment_balance => certificate.price,
          :line_item_id => certificate.id,
          :account => self.sale.account)
      end
    end
  end
end
```

Effects on Shipping

As previously mentioned, gift certificates can affect the shipping for an order. If the order contains only gift certificates, you don’t even need a physical address, as you won’t be delivering anything. Obviously you also don’t need to add a shipping cost. As a result, you’ll need to have a slightly different page flow when someone attempts to complete an order that has only gift certificates. You can skip the address collection, and your final check before allowing the customer to complete the order should omit the check for a valid address and shipping method. You need a simple boolean:

```
def needs_address?
  self.line_items.reject {|item|
    item.product.is_a?(GiftCertificate) &&
    item.product_options.collect(&:weight).sum == 0
  }.any?
end
```

Now you can easily check whether the user’s cart needs to have a valid address associated with it. Once you know you don’t need a real address and that you don’t need the user to select a real shipping method, you can fake it with a virtual shipping method. We call ours “Digital Delivery”. Of course, this `needs_address?` function can be modified to

check for other items in your catalog that are also delivered electronically, such as e-books, music and video files, etc.

You might be wondering what that 4th line in the snippet is about, the one that sums up the weight of product options. If you glance back at the code snippets from the Product Options section, you can see that product options can have their own price and weight. Gift wrap may add an additional \$3.00 to the order, for example. In this case, gift certificates have a product option peculiar to them: a gift card. A customer can choose the gift card product option to not only get the gift certificate redemption code via email, but can also choose to get a physical card imprinted with the redemption code mailed to them. This adds both a cost and a weight to a gift certificate, and now requires a shipping address for an item that didn't previously require it. Having a gift card alternative for gift certificates was actually the motivating factor for having the product option functionality in our application.