

HOW TO SELL VIRTUAL CURRENCY - QUICKSTART

Firstly, create a new product in WooCommerce just like you create any other product.

To enable your currency product for sale and to have it integrate with your Unity games you then need to follow a few additional steps while creating your WooCommerce Product:

1. The product must be marked as a simple product
2. The product must be marked as virtual and non-downloadable
3. The product SKU must be formatted for that specific product type (see below).
4. Done.

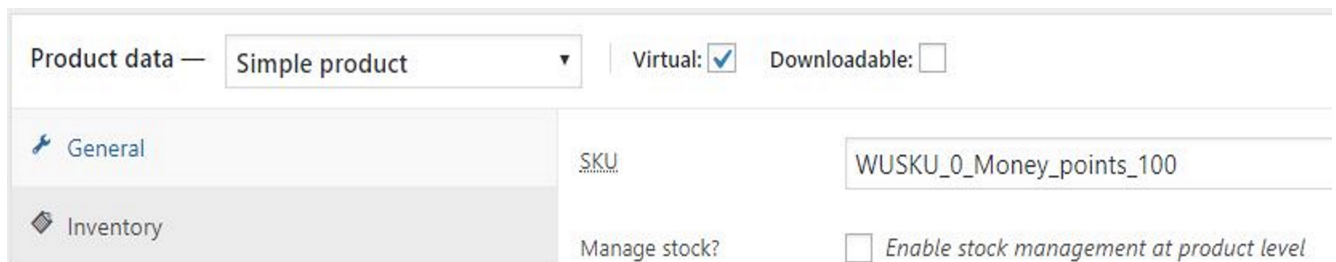
Follow the instructions above and give your virtual currency an SKU that follows the following format:

WUSKU_{GameID}_MONEY_{CurrencyName}_{Qty}

Example SKUs:

WUSKU_1_MONEY_Gold

WUSKU_1_MONEY_Gold_200



Product data — Simple product

Virtual: ☒ Downloadable: ☐

General

SKU: WUSKU_0_Money_points_100

Inventory

Manage stock? ☐ Enable stock management at product level

SELLING IN GAME CONTENT - ADDITIONAL INFO

All products that you want to sell in-game must have an SKU that starts with at least three basic parts:

1. The first part must always be uppercase "WUSKU"
2. The second part is the game ID that product is created for or 0 if the product can be shared between all the games on your website
3. Lastly, it is very important to include some type of indicator to tell you what type of purchase this is or what asset type it is for. More info on this later.

More parts may be added as required / determined by one's purchase function when it detects this sale.

Since there may be more than one function listening for item sales one needs a way to determine which function should handle the sale of any one particular item. This is the purpose of the third part of the SKU.

Let's use the virtual currency plugin as an example. When one sells currency one will most likely sell it in bundles of 50, 100, 1000 or 10000 etc and since *Bridge : Money* allows one to have as many virtual currencies as one wishes, one gets to specify what currency is being sold, as well as and how large the bundle is, right inside the SKU. If one needs more info about the purchase, simply add it...

PRO TIP: Since customers are able to add multiple coins bundles to their cart one needs to be able to multiply the currency bundle size with the number of bundles in the cart in order to determine how many coins to ACTUALLY award customers. For this reason I suggest one specifies the number of coins in the bundle as part of the SKU itself. This gives one easy and direct access to both values.

On the other hand, when one wants to sell a game, for instance, all one wants to know is that the buyer is buying the game with id 8, no additional info is really needed in this case. As stated at the start of this section, an SKU requires 3 parts only.

on

his

00,
cify

ard
ves

ne
y

but should one need extra info like the bundle size in the previous example, one has the option to add any additional field to satisfy one's requirements. Let's look at an example of a transaction that needs extra info as well as one that does not:

Example SKU for the game with Id 8: WUSKU_8_GAME

Example SKU for a bundle of 10 Gold: WUSKU_8_MONEY_Gold_10

This third parameter exists to give the various functions that handle purchases a way to know whether or not to process one particular transaction/product. In this case (and the recommended workflow) the virtual currency purchase function tests the third part of the SKU right at the start of the function and if the function finds the value is set to anything other than "MONEY" it exits immediately. (See the next section for more info on custom product types)

I will be releasing more functions in future enabling more product types and will explain their respective SKU requirements inside each script as I do so. You are not required to wait for me to create any custom product type handler for you, though the code is in place to allow anyone to specify their own product type handler functions and sell anything they want...

ADVANCED INFO FOR ADVANCED USERS

NOTE: Teaching you how to create or load php code for WordPress falls outside the scope of this product. This is an advanced use scenario and requires existing knowledge of both php and WordPress plugin management.

We will be releasing more functions over time to handle more product types... but should you want to sell something of your own before we get around to making the relevant plugin or function for that item, we have made it very easy for anyone with any level of PHP experience to make their own custom functions and sell their own custom content.

NOTE: Probably the biggest problem you will face will be how to link the purchase to the buyer's account. To aid with this we highly recommend the [Bridge : Data](#) asset. You are free to save this info in a multitude of ways (as many ways as you can think of, to be exact) so the Bridge : Data asset is by no means mandatory, just highly recommended purely for the simplicity it offers. It will save your data to the correct account and fetch it back from there again afterwards, back in Unity. This type of thing (saving custom data and fetching it back in Unity) was exactly why the [Bridge : Data](#) asset was created in the first place

In order to handle custom item purchases you need to first create then load a new script. This script needs to do the following in order to be considered compatible with the Bridge : Money asset:

1. You need to register your function with the *Bridge : Money* 'wub_purchase' action :

```
add_action('wub_purchase','MyFunction');
```

Where 'MyFunction' is the name of your custom function to handle the purchase. NOTE: All custom function names must be universally unique, as per the WordPress function name requirements.

2. Next you need to create the actual MyFunction function you just specified and give it exactly one argument (to keep things uniform, we recommend you call it \$args):

```
function MyFunction($args)
```

3. Finally, test the third parameter to see if the current product is relevant to your function so you can exit if not. Here is how it was done in the virtual currency function:

```
if(strtolower($args['type'])!="money")  
return;
```

Those are all the steps one needs to take in preparation for selling custom content. From here on forward what your function does is entirely up to you. We try to provide it with as much info as possible to give the developer as much assistance as possible.

ds

any

than

ts

gh...

ce

our

h

ys

ely

s

a

s

p

\$args (or whatever you called it) is an associative array that contains 7 values to help you along.

uid - The id of the person who bought this product. Matches the value in the WordPress *users* table

gid - The ID of the game this item was created for

type - An identifier to help you determine if your function should process this item

fields - If your SKU contained more than 3 parts this numeric array contains the remaining parts

qty - The cart quantity specifying how many of this particular product was bought

data - The data WooCommerce collected for this item including meta data

product - Just in case you still need more, this is the entire WooCommerce PRODUCT thus giving you access to absolutely everything there is to know about this product

As you can see, data is provided three times at different levels of ease of access. The most common items are available directly from *\$args* while others are nestled deep within. Knowing this, you might choose to always have your SKU be only parts long and keep all other info in the meta data. That will definitely give you a much better looking SKU but at a cost of relevant data being slightly harder to get to... but definitely possible. Knowledge of the WooCommerce API is required for this, though

Complete example (SKU: WUSKU_0_LOYALTY):

```
add_action('wub_purchase', 'AddOneLoyaltyPoint');
function AddOneLoyaltyPoint($args)
{
    if (strtolower($args['type']) != "loyalty") return;
    $keyname = "{$args['id']}_LoyaltyPoints";
    $user_id = $args['uid'];

    $count = get_user_meta($user_id, $keyname, true) ;
    If ($count === false) $count = 1; else $count = intval( $count ) + 1;
    update_user_meta($user_id, $keyname, $count);
}
```

y 3
f the
r