

# HOW TO SELL IN-GAME CONTENT FROM YOUR WEBSITE

## OVERVIEW

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 must follow the SKU format specific to that product type.
4. Done.

## HOW TO SELL VIRTUAL CURRENCY - QUICKSTART

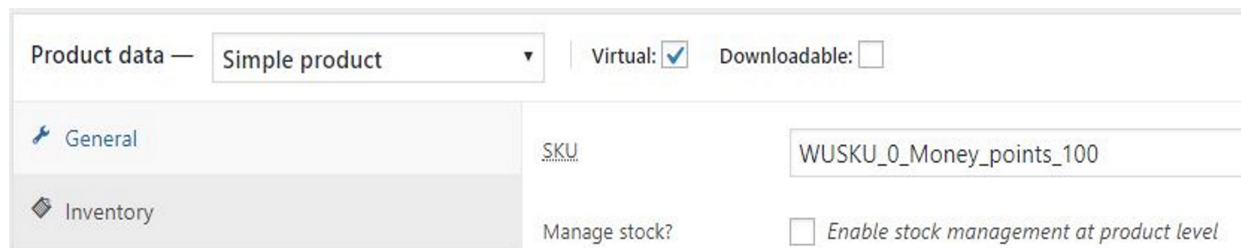
Follow the instructions above and give your virtual currency an SKU that follows the following format:

WUSKU\_{GameID}\_MONEY\_{CurrencyName}\_{Qty}

### Example SKUs:

WUSKU\_1\_MONEY\_Gold

WUSKU\_1\_MONEY\_Gold\_200



The screenshot shows the 'Product data' section of the WooCommerce admin interface. The 'Product type' dropdown is set to 'Simple product'. The 'Virtual' checkbox is checked, and the 'Downloadable' checkbox is unchecked. Below this, the 'SKU' field is populated with 'WUSKU\_0\_Money\_points\_100'. The 'Inventory' section is also visible, with the 'Manage stock?' checkbox unchecked and the label 'Enable stock management at product level'.

## SELLING IN GAME CONTENT - ADDITIONAL INFO

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

1. The first part must always be "WUSKU" and must be uppercase.
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 this is for. More info on this later.

You can add more parts if you like. This will change based on what you want to do once you detect this product was sold. This is where the third parameter comes into play.

Since there may be more than one function listening for items that was sold you need to determine whether or not this item should be processed or ignored by any one particular function.

Let's use the virtual currency plugin as an example. When I sell currency I will most likely sell them in bundles of 50, 100, 1000 or 10000 etc and since WUMoney allows you to have as many virtual currencies as you want, I want to specify what currency I am selling and how large the bundle is as part of the SKU.

**NOTE:** A customer can add multiple products to his cart so you can't rely on the qty of the order to determine how many items to give the customer. That is why I specify the number of coins as part of the SKU. In the case of the Virtual Currencies function I multiply the order quantity by the product's quantity to determine how much currency was bought

Since my SKU has a currency name and an optional quantity attached to it (defaults to 1 if not specified) I want the WUMoney plugin's function to handle this purchase and thus I set the third parameter to MONEY. The WUMoney virtual purchase function tests the third part to see if it is MONEY and exits immediately if not.

However, when you want to sell a game, for instance, All you want to know is that you are selling the game with id 8 so in that case you just have to specify the SKU like this:

I will be releasing more functions in future enabling you to sell more items and will explain their respective SKU requirements inside each asset.

## **ADVANCED INFO FOR ADVANCED USERS**

I will be releasing more functions over time but if you want to sell something of your own before I get around to making the relevant plugin or function, I have made it very easy for you to make your own functions to sell your own content.

**NOTE:** Your biggest problem will be how to save the data but if you have the WUData kit that might be of help to you. Alternatively, how you store and later bring the data into Unity is up to you to decide.

In order to handle custom item purchases you need to create and load a new script and do the following three things first:

**NOTE:** Teaching you how to create php scripts or how to load them inside of WordPress falls outside the scope of this product. This is an advance use scenario so you will need to know what you are doing if you want to attempt this

1. You need to register your function with my action by doing the following:

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

Where 'MyFunction' is the name of a function you will create to handle the purchase. All these function names must be unique as per the standard WordPress requirement.

2. Next you need to create the MyFunction function (or whatever you called yours) and give it one argument. For uniformity sake, call it \$args. Here is the function created for selling virtual currency

```
function wuss_money_purchase($args)
```

3. Finally, test the third parameter to see if the current product is relevant to your function and 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 you need to take in preparation for selling your content. From here on forward what you do is entirely up to you so I try to provide you with as much info as possible.

\$args is an array that contains 7 values to help you along.

uid - The id of the person who bought this product

gid - The game id 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 will contain the rest

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

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

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

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 3 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 the relevant data being slightly harder to get to... but definitely possible. Knowledge of the WooCommerce API is required for this, though