Custom Items

By Goinza

This guide will focus on the creation of custom items. Before we begin, we need to explain what a custom item is: each item has an Item Type, which determines the effect of that item. For example, a HP Recovery item restores the HP of a unit, a Class Change item changes the class of a unit, a Learn Skill item gives the unit a new skill, and so on.
In the case of the Custom item type, there is not a specific effect. Instead, you specify a custom keyword and code the effect with a plugin. As an example for this guide, we will create a custom item that increases the growth rate of the targeted unit, showing all the steps necessary to do this. Keep in mind that the full code can be accessed in the growth-bonus.js file.

Custom Keyword

Before we start coding, we need to specify the keyword that we will use for this new item type. It is important to avoid using the same name as other item types from other plugins. This keyword is a simple string, like for example "GrowthBonus".
We can create a variable and store the value of the keyword there. While this is not necessary, it is a good idea because that way you can use the variable instead of writing the keyword in every line of code that requires it, so whenever you want to change the name of the keyword, you only need to change the value of the variable.

```
var GROWTH_ITEM_KEYWORD = "GrowthBonus";
```

The six objects used for items

To create the code for a new type of item, we need to create 6 objects: each one of them will inherit the data from one of the "base" objects that are shared among all items:

- BaseItemSelection: handles the logic behind the selection of targets of the item.
- BaseItemUse: when the item is used, the code from this object is executed.
- BaseItemInfo: the information that will be displayed when you check the data of an item in a menu.
- BaseItemPotency: shows the scale of the item's effect. This is not applicable for all items. For example, in a HP Recovery item, it shows how much healing the item will do, but in an Unlock item it won't show up anything.
- BaseItemAvailability: restricts which units can use the object.
- BaseItemAI: the logic used by the AI to determine when and if they should use the item. Some of the item types don't have AI logic so the enemy units won't use those items.

Once we identify these objects, we can start creating our own objects. At first, we will make them empty, and we'll add the functions later.

```
var GrowthItemSelection = defineObject(BaseItemSelection, {

})

var GrowthItemUse = defineObject(BaseItemUse, {

})

var GrowthItemInfo = defineObject(BaseItemInfo, {

})

var GrowthItemPotency = defineObject(BaseItemPotency, {

})

var GrowthItemAvailability = defineObject(BaseItemAvailability, {

})

var GrowthItemAI = defineObject(BaseItemAI, {

})
```

ItemPackageControl Functions

In addition to creating the objects, we also need to overwrite some functions to let know the game that if an item with the custom keyword "GrowthBonus" appears, it should be assigned these group of objects. All of these functions come from the object ItemPackageControl, which is located in the file item-base.js.

For example, the function getCustomItemSelectionObject returns the selection object associated with the current item. In this case, the object is GrowthItemSelection.

```
    var alias1 = ItemPackageControl.getCustomItemSelectionObject;
    ItemPackageControl.getCustomItemSelectionObject = function(item,
keyword) {
        if (keyword == GROWTH_ITEM_KEYWORD) {
            return GrowthItemSelection;
        }

        return alias1.call(this, item, keyword);
    }
```

As you can see in the code, if the item's keyword is equal to the string stored in our variable, then the object GrowthItemSelection is used. Else, it calls alias1 and delegates the job to other instances of the function, either from the default code or from other plugins.
This needs to be done for each object. In our example of the growth item, the assignment should be like this:

Function	Object
getCustomItemSelectionObject	GrowthItemSelection
getCustomItemUseObject	GrowthItemUse
getCustomItemInfoObject	GrowthItemInfo
getCustomItemPotencyObject	GrowthItemPotency
getCustomItemAvailabilityObject	GrowthItemAvailability
getCustomAIObject	GrowthItemAI

With the current code, we now have a custom item that is possible to use, but it doesn't have any effect yet. So we need to delve into each object and start adding code.

GrowthItemSelection

The parent object BaseItemSelection already has code to deal with the targeting system. This means that if the item only needs to target units, then the base code is enough. The growth item we are creating doesn't need any extra code for this, so the GrowthItemSelection object will be empty.

GrowthItemUse

For this object, we need to overwrite two functions. The first function is called enterMainUseCycle, which basically activates the use of the item. Usually, after executing this function, you "enter" the object cycle and it starts calling the functions moveMainUseCycle and drawMainUseCycle once every frame. But to make this item simpler, we will ignore those two functions. What we will do is when the function enterMainUseCycle is called, it will call the mainAction function, which will apply the item's effect on the target unit.

```
    enterMainUseCycle: function(itemUseParent) {
        this._itemUseParent = itemUseParent; //Store the object to use later
        this.mainAction(); //Main method of the object
        return EnterResult.OK;
    },
```

While this function is relatively simple with just three lines of code, the mainAction function can be considered the most complex function of this plugin, due to calling some functions that you might not be familiar with. So instead of showing the code here, it is better for you to read the actual plugin on the file growth-bonus.js, which has comments explaining every line of code of the function.

GrowthItemInfo

As we explained before, this object handles the window that shows the details of the item. For the growth bonus type of item, it will look like this:

To edit this window, we need to use two functions: getInfoPartsCount, which determines the amount of rows of text that the window will have, and drawItemInfoCycle, which sets all the information on the window.

For the getInfoPartsCount function, the code is extremely simple: we always want two rows: one for the item type name, and the other for the bonus and range of the item.

```
    getInfoPartsCount: function() {
        return 2; //Amount of rows used on the item's window
    }
```

For the drawItemInfoCycle function, we need to use more code: basically, it comes down to the use of three functions:

- ItemInfoRenderer.drawKeyword: draws a string with the yellow color which highlights it.
- NumberRenderer.drawRightNumber: draws a number. In this case it will be used for the bonus value.

- this.drawRange: the “this” word means the current object, but GrowthItemInfo doesn’t have that function by itself, it inherited the function from BaseItemInfo. This function is shared by most types of items and it is used to draw the range of items.

Using all this functions, we can write the code for the function drawItemInfoCycle.

```
    drawItemInfoCycle: function(x, y) {
        ItemInfoRenderer.drawKeyword(x, y, this.getItemTypeName("Growth"));
        y += ItemInfoRenderer.getSpaceY();

        ItemInfoRenderer.drawKeyword(x, y, "%");
        x += ItemInfoRenderer.getSpaceX();
        NumberRenderer.drawRightNumber(x, y, this._item.custom.bonus);

        x += 40;
        this.drawRange(x, y, this._item.getRangeValue(),
this._item.getRangeType());
    }
```

Notice that there are more functions than the ones mentioned above: getSpaceX and getSpaceY return a specific number, and they are used to separate the coordinates of every string or number written. And getItemTypeName is a function that returns the name of the item type followed by the “Staff” or “Item” words, depending if the item is a staff or a normal item, respectively.

GrowthItemPotency

This object is used by only a couple of item types. It shows detailed information on the action of a unit using the item on another unit. For example, for a HP Recovery item type it will show how much healing will be done on the target unit.
In the case of our custom item, we want to show the bonus growth, similar to how it was done in the GrowthItemInfo object.

To do this, we need to use some simple functions: setPosMenuData, which stores the bonus growth in the object for later use and getKeywordName, which returns a small string that can be used to determine the type of bonus we are using (in this case, the string is '%' to represent percentage).
In addition to those two functions, we also need to use drawPosMenuData.

```
    drawPosMenuData: function(x, y, textui) {
        var font = textui.getFont();

        TextRenderer.drawKeywordText(x, y, this.getKeywordName(), -1,
ColorValue.KEYWORD, font);
        NumberRenderer.drawNumber(x + 65, y, this._value);
    }
```

This function does the main work of this object, which is rendering the data that we show in the window of the screenshot above. The function drawKeywordText handles the text pulled from the getKeywordName function and the drawNumber function handles the drawing of the growth bonus value.
Note that the variable this._value was already defined in the function setPosMenuData.

GrowthItemAvailability

Most of the logic of this object is already defined in the BaseItemAvailability object. We are only interested in the isItemAllowed function. This function returns true if the unit can use the item, and false if it can't.

```
    isItemAllowed: function(unit, targetUnit, item) {
        return targetUnit.custom.usedGrowth == null;
    }
```

As you can see, the function is very simple. It checked that the unit doesn't have the custom parameter "usedGrowth", which is defined in a unit whenever that unit uses a custom GrowthBonus item. This is done like this to prevent a unit to use more than one time an item of the GrowthBonus type.
If you want to see how this parameter is used in the example plugin, check the last line on the function mainAction of the object GrowthItemUse.

GrowthItemAI

This type of item was created with the idea that only the player can use it. Therefore, the GrowthItemAI is empty. This way, AI units won't be able to use any item of the GrowthBonus type.

Conclusions

With this guide you should be able to create your own custom item types.
In conclusion, for each item type that you want to create, you need to define a keyword, modify 6 functions from ItemPackageControl to make them check for the new keyword, and finally create your own 6 objects that will be used by those functions.
Remember to check the growth-bonus.js as a reference. While most of the important code was explained here, that plugin contains the full code used to make the GrowthBonus custom item type, and it also includes additional comments.