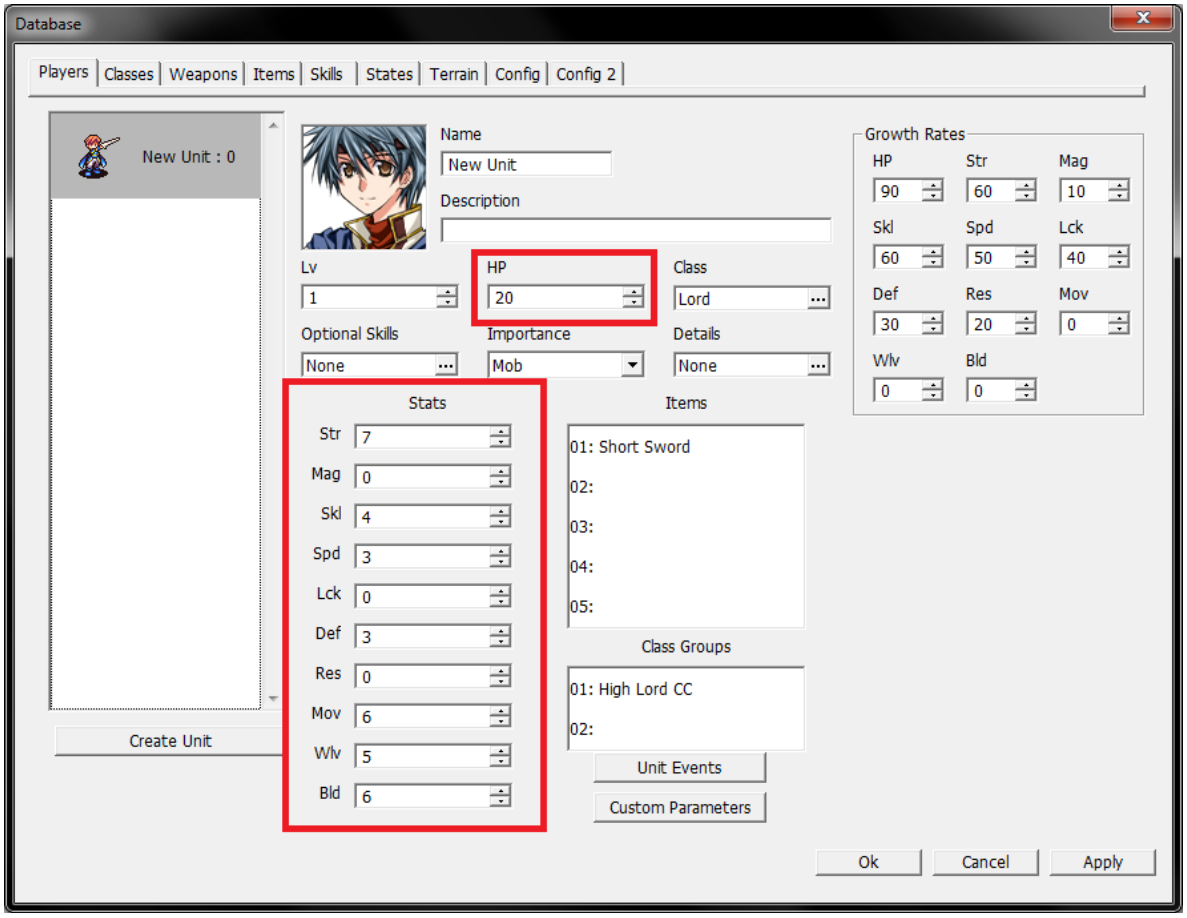
Creating a New Stat

By Goinza

This guide will explain how to create a new unit stat and use it for combat, among other things. First, we need to determine what a unit stat is. If you go to the Database window and select the Players tab, you can see the stats of each unit that is in that tab.



New Unit : 0

Name: New Unit
Description:
Lv: 1
HP: 20
Class: Lord
Optional Skills: None
Importance: Mob
Details: None

Stats

Str	7
Mag	0
Skl	4
Spd	3
Lck	0
Def	3
Res	0
Mov	6
Wlv	5
Bld	6

Items
01: Short Sword
02:
03:
04:
05:

Class Groups
01: High Lord CC
02:

Growth Rates

HP	90
Str	60
Mag	10
Skl	60
Spd	50
Lck	40
Def	30
Res	20
Mov	0
Wlv	0
Bld	0

In here, you can see all the unit's stats: Max HP, Strength, Magic, Skill, Speed, Luck, Defense, Resistance, Movement, Weapon Level and Build.
Inside the code, each of these units is handled by a different object, and each of those objects is a child of the base object called BaseUnitParameter. For example, the code for the Defense stat is this:

```
UnitParameter.DEF = defineObject(BaseUnitParameter,
{
    getParameterType: function() {
        return ParamType.DEF;
    },

    getSignal: function() {
        return 'def';
    }
}
);
```
As you can see, this object seems too simple, it only has two functions. That is because most of the work is handled by the object BaseUnitParameter, which can be seen on the file singleton-paramgroup.js line 144.

New Stat Object

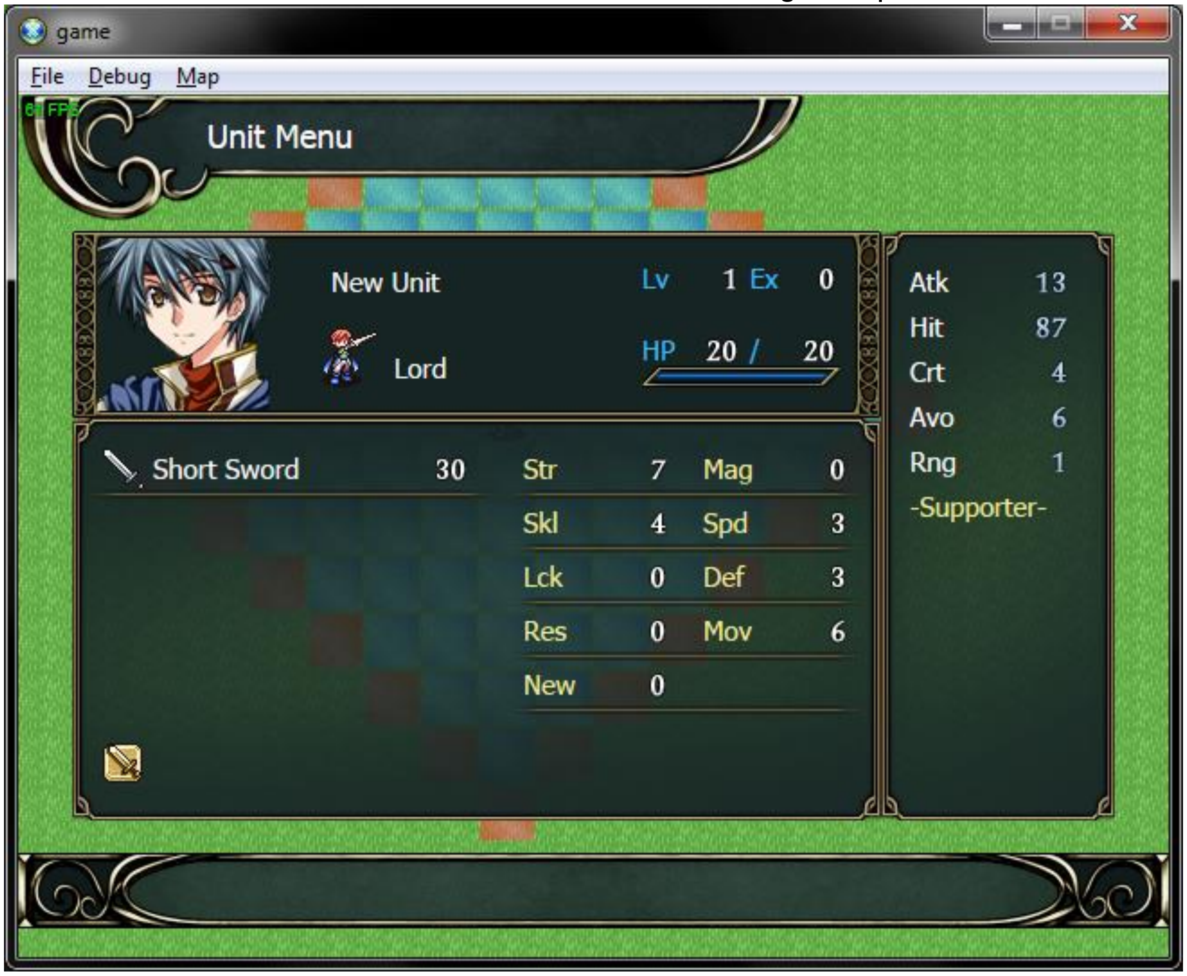
Now, let’s try to make a new stat object that inherits from BaseUnitParameter.
We will begin by making the simplest object possible. For now, we will only add two functions, called getParameterName and getParameterType, which determine the name and type of the stat. If this first function is not overwritten, the game will crash, and the second is necessary to obtain the value of the stat on any unit from anywhere in the code, so make sure to at least overwrite these functions before testing the game.
```
ParamType.NEWSTAT = 100;

var NewStat = defineObject(BaseUnitParameter, {

    getParameterName: function() {
        return "New";
    },

    getParameterType: function() {
        return ParamType.NEWSTAT;
    }
})
```
Note that ParamType is the object that stores all the types of stats. This can be seen in constants-enumeratedtype.js line 184. In this case, we are adding a new value called NEWSTAT and we give it an arbitrary number. The value doesn’t matter, as long as no other stat shares the same value.
This code by itself won’t do anything. To make it visible in the game, we need to change the function ParamGroup._configureUnitParameters. This is a function that is given an empty array and fills it with all the stat objects of the game. We are going to extend it to make it so it also adds the new stat object we created.

```
var alias1 = ParamGroup._configureUnitParameters;
ParamGroup._configureUnitParameters = function(groupArray) {
    alias1.call(this, groupArray);
    groupArray.appendObject(NewStat);
}
```
Note that, even though we added a new stat, that change will never be reflected in the engine editor. You can only check the changes during real gameplay. So for this, we will check the Unit Menu Screen of a unit during a map test.



New Unit Lv 1 Ex 0
Lord HP 20 / 20

Short Sword 30

Str	7	Mag	0
Skl	4	Spd	3
Lck	0	Def	3
Res	0	Mov	6
New	0		

Atk	13
Hit	87
Crt	4
Avo	6
Rng	1

-Supporter-

As result of our new code, the stat is now visible in the screen, but it is always at 0, and we currently have no way to modify that. So now we will start to overwrite functions of BaseUnitParameter to add more functionality to the stat.

Using basic functions from BaseUnitParameter

To add functionally to the new stat, we have to overwrite some of the functions of BaseUnitParameter on the new stat object.

getUnitValue

First, we will use the getUnitValue function, which returns the current value of the stat for a specific unit. Usually, the value is taken from the Database window, like the ones shown in the first screenshot, but that is not possible for new stats. Instead, we will use a custom parameter. For this example, we can use the parameter “newStat” which holds the current value of the stat for that particular unit. The parameter will look like this:

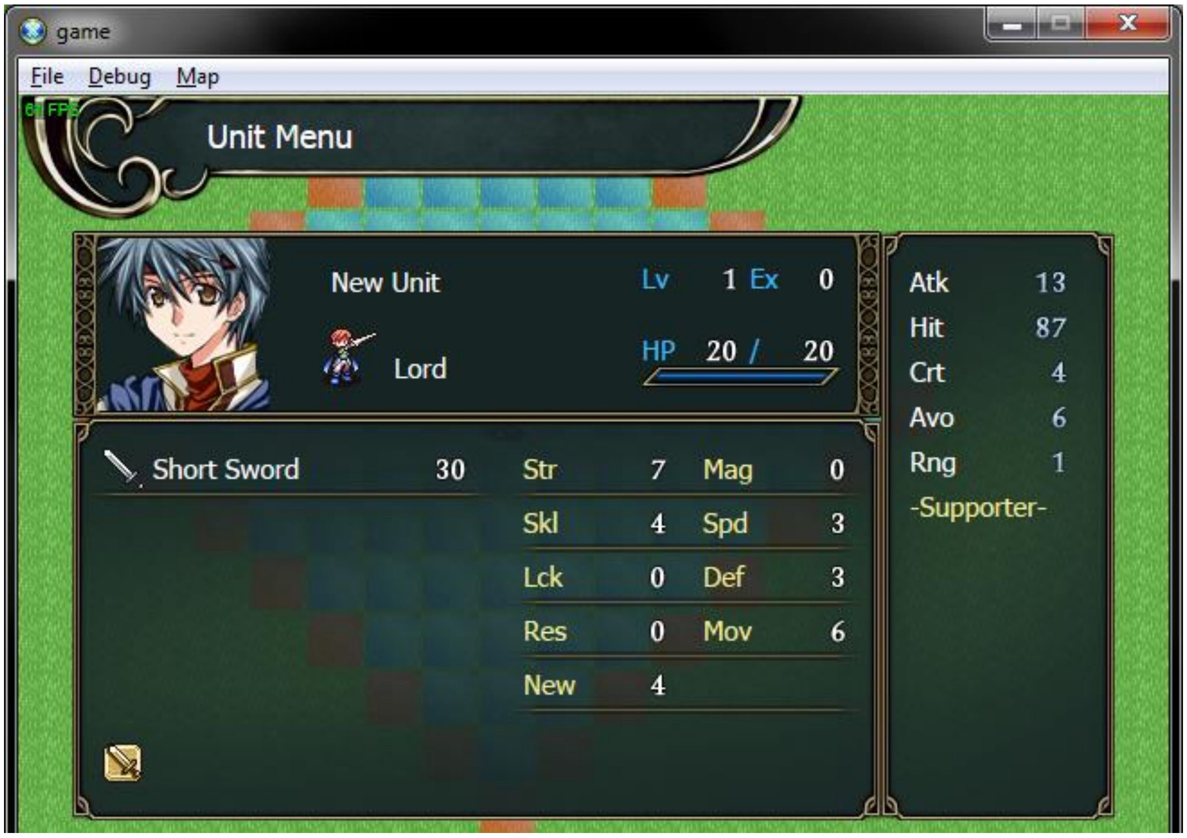
{newStat: 4}

Now that the unit has the custom parameter, we can use it in the getUnitValue function.

```
    getUnitValue: function(unit) {
        var value = 0; //Default value
        if (unit.custom.newStat != null) {
            //Checks that the parameter exists to avoid crashes
            value = unit.custom.newStat;
        }

        return value;
    }
```

Now the unit will have a value for the new stat.



New Unit Lv 1 Ex 0
Lord HP 20 / 20

Short Sword 30

Str	7	Mag	0
Skl	4	Spd	3
Lck	0	Def	3
Res	0	Mov	6
New	4		

Atk	13
Hit	87
Crt	4
Avo	6
Rng	1

-Supporter-

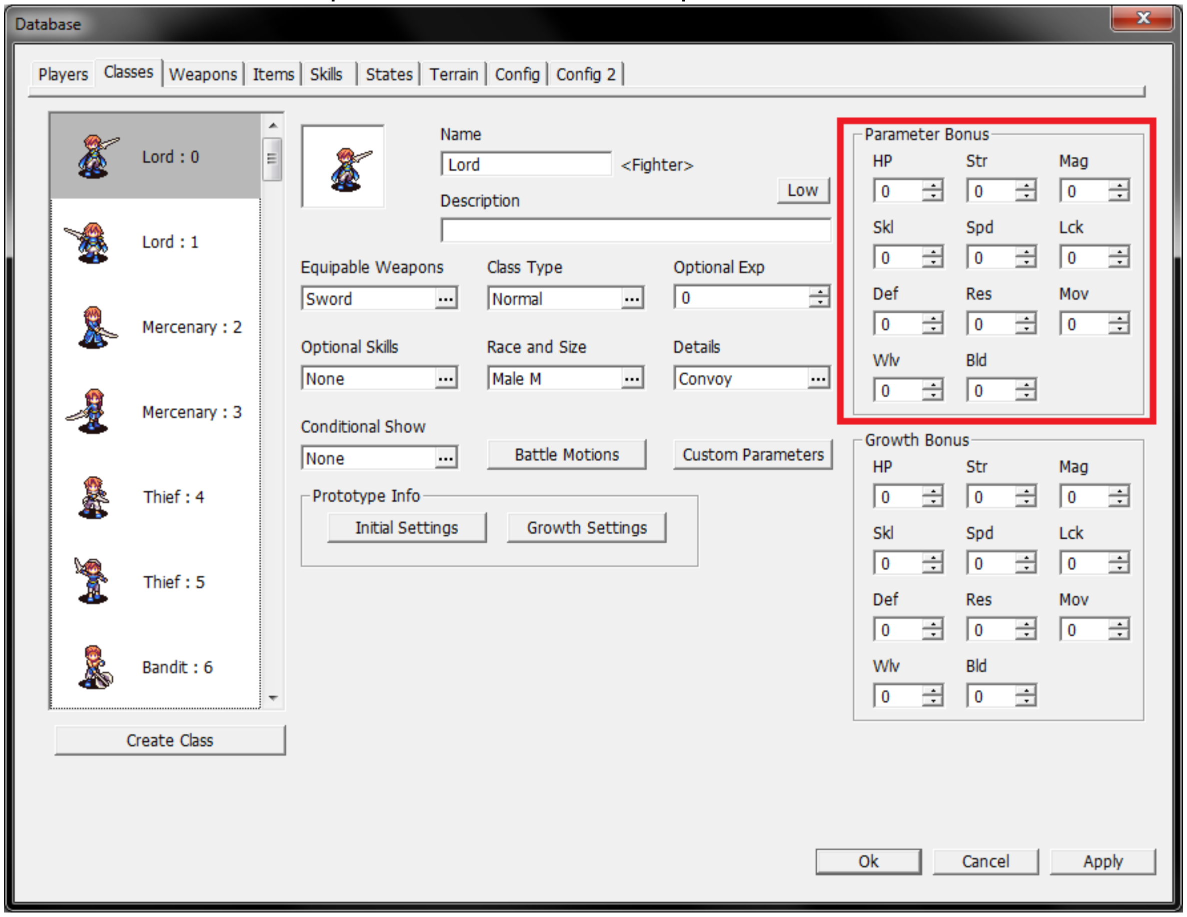
setUnitValue

Similar to getUnitValue, but instead of returning the current value, it changes the value to a new one. This implies changing the value of the unit's custom parameter.

```
    setUnitValue: function(unit, value) {
        unit.custom.newStat = value;
    }
```

getParameterBonus

This function takes a certain object related to the unit and adds a bonus to the stat of that unit. For example, if the object is the unit's current class, it adds the bonus determined by that class. By default, normal stats use the Parameter Bonus, which is available in different places, like classes, weapons, items, etc.



Name: Lord <Fighter> Low
Description:
Equipable Weapons: Sword
Class Type: Normal
Optional Exp: 0
Optional Skills: None
Race and Size: Male M
Details: Convoy
Conditional Show: None

Parameter Bonus

HP	Str	Mag
0	0	0
Skl	**Spd**	**Lck**
0	0	0
Def	**Res**	**Mov**
0	0	0
Wlv	**Bld**	
0	0	

Growth Bonus

HP	Str	Mag
0	0	0
Skl	**Spd**	**Lck**
0	0	0
Def	**Res**	**Mov**
0	0	0
Wlv	**Bld**	
0	0	

To do this with a new stat, we will need another custom parameter. In this example, we will call it "newStatBonus". This parameter can be added to any object that has the Parameter Bonus option and it will be used by the getParameterBonus function.

```
    getParameterBonus: function(obj) {
        var value = 0; //Default value
        if (obj.custom.newStatBonus != null) {
            //Checks that the parameter exists to avoid crashes
            value = obj.custom.newStatBonus;
        }

        return value;
    }
```

getGrowthBonus

This function handles the growth value of a unit. This will need another custom parameter, which we will call “newStatGrowth”, and can be used in any object that has the Growth Rates or Growth Bonus options. This includes units, classes, weapons, etc.

```
    getGrowthBonus: function(obj) {
        var value = 0; //Default value
        if (obj.custom.newStatGrowth != null) {
            //Checks that the parameter exists to avoid crashes
            value = obj.custom.newStatGrowth;
        }

        return value;
    }
```

getDopingParameter

This function handles the bonus value obtained by items of the Stat Boosting type. Usually, you use the Item Effects window to determine which stats are boosted with the use of this item. But for the new stat, we are going to use the custom parameter “newStatDoping”.

```
    getDopingParameter: function(obj) {
        var value = 0; //Default value
        if (obj.custom.newStatDoping != null) {
            value = obj.custom.newStatDoping;
        }

        return value;
    }
```

getMaxValue

This function determines the max value possible for the stat. For normal stats, it can be defined by either a global value in the Config2 tab or have each class have its own max value. For this example, we are going to do the class approach. This means that each class will need a custom parameter, which we will call "newStatMax".

```
    getMaxValue: function(unit) {
        var value = 0; //Default value
        var unitClass = unit.getClass();
        if (unitClass.custom.newStatMax != null) {
            value = unitClass.custom.newStatMax;
        }

        return value;
    }
```

Using the new stat

Now that the basic functions are complete, we need to create a function that can fetch the current value of the stat on a unit, taking into account the bonus from classes, weapons, states, etc.
In the case of normal stats, the objects RealBonus and ParamBonus are used. For example, to get the Magic stat of a unit, you call the function RealBonus.getMag, which then calls ParamBonus.getMag and finally it goes to the function ParamBonus.getBonus. So for our new stat, we are going to create a new function that calls the ParamBonus.getBonus function.

```
var NewStatBonus = {
    getNewStat: function(unit) {
        return ParamBonus.getBonus(unit, ParamType.NEWSTAT);
    }
}
```

Note that for this new stat I created a new object. This is not the only way to solve the problem. You can also choose to store the function into the variable without being nested into an object, or add the function into an object that already exists in the engine, like for example ParamBonus.
Now that we have this function, we can get the value from anywhere in the code. Let's use this to modify the formula to calculate the attack value of a unit. By default, it is Str + weapon's power if it is a physical attack and Mag + weapon's power if it is a magical attack. We are going to use that formula and add the new stat's value to the result, no matter the type of attack. The function that handles this is AbilityCalculator.getPower located at singleton-calculator.js line 3

```
//The new stat adds more damage to any attack
var alias2 = AbilityCalculator.getPower;
AbilityCalculator.getPower = function(unit, weapon) {
    var power = alias2.call(this, unit, weapon);
    var newStatValue = NewStatBonus.getNewStat(unit);

    return power + newStatValue;
}
```

In conclusion, for each thing you want to add to the engine about the new stat, you need to find the appropriate function and modify it to take into the account the new stat using the NewStatBonus.getNewStat function.

Renderable Parameters

There is an alternative to the display of stats that are used in the engine. By default, all stats are shown as numerical values, but it is possible to show other types of values, including words and icons. Internally, the stat will still be a number, but we can map each number to other things.
For example, we can create a stat called Affinity that shows the magical affinity that the unit has with a specific element. In this case, each number will be associated with an element: 0 for Ice, 1 for Water, 2 for Earth, 3 for Fire, 4 for Air and 5 for Thunder.
To make this example simple, we will assume that there aren't any bonus from classes, boosting items or growth rates. Each unit gets a fixed value from a custom parameter and that can't be changed. Because this stat doesn't have any impact on the gameplay mechanics, we don't need to make a function to fetch the current value of the unit, we just need enough code to make the stat displayable.
First, we need to create a new object that inherits from BaseUnitParameter, and then use the functions getUnitValue, which will get the unit custom parameter called "aff", and getParameterName, which returns the stat's name. These two functions will be used the same way as explained with the example above.

```
var Affinity = defineObject(BaseUnitParameter, {

    getUnitValue: function(unit) {
        var value = 0; //Default value
        if (unit.custom.aff != null) {
            //Checks that the parameter exists to avoid crashes
            value = unit.custom.aff;
        }

        return value;
    },

    getParameterName: function() {
        return "Aff";
    }
}
```

Now that we have a displayable value, we need to add other functions to this object in order to make it "renderable", which means that you can display the stat with anything you want, instead of plain numbers.
The functions we need are isParameterRenderable and drawUnitParameter. The first returns a boolean, which is false by default. We need to override it to make it

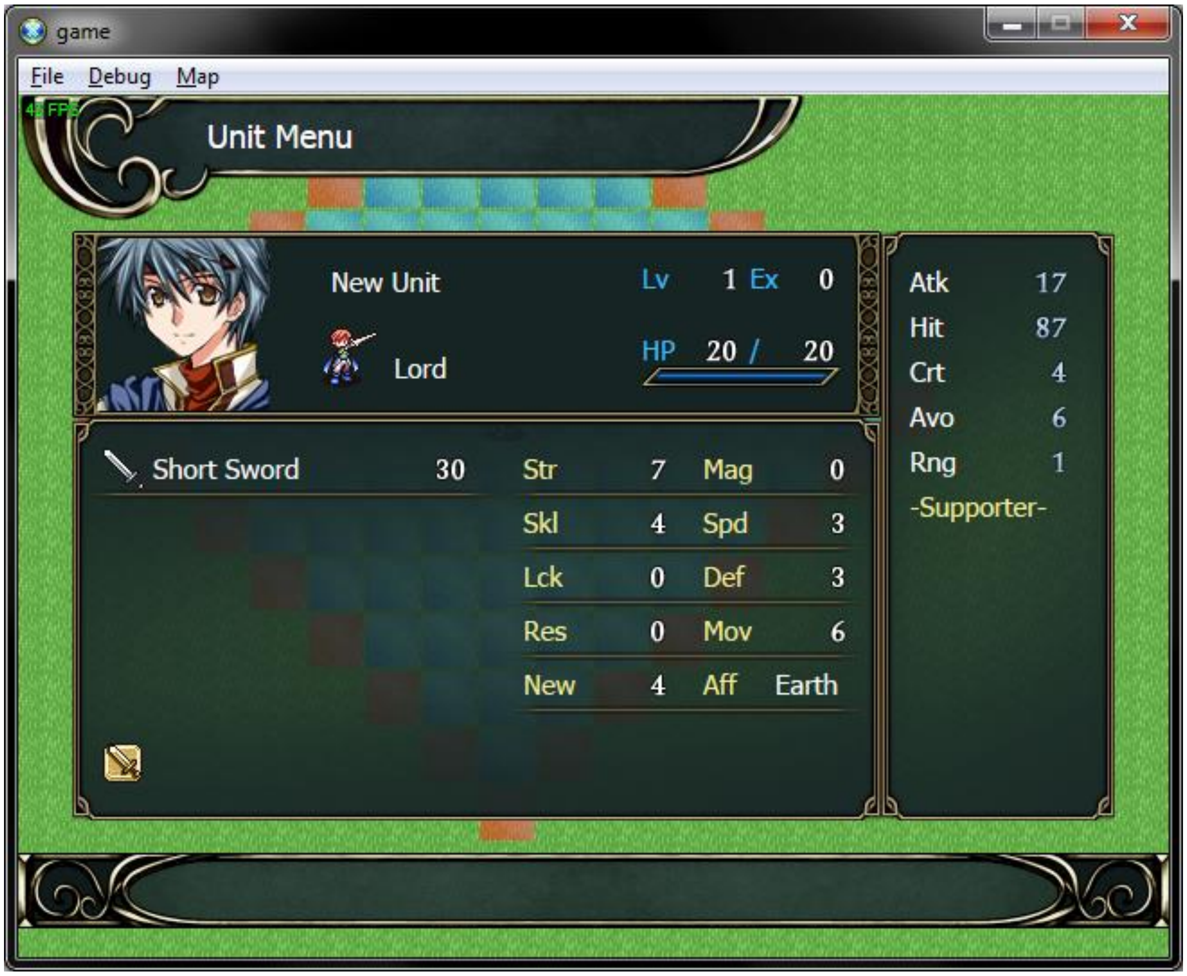
return true. And the second is the function that renders the new data. In this case, we are going to draw one word by using the function drawKeywordText.

```
    isParameterRenderable: function() {
        return true;
    },

    drawUnitParameter: function(x, y, statusEntry, isSelect) {
        var textui = statusEntry.textui;
        var color = textui.getColor();
        var font = textui.getFont();
        var value = statusEntry.param; //Current value of the stat
        var text = ""; //Default value
        switch (value) {
            case 0:
                text = "Ice";
                break;
            case 1:
                text = "Water";
                break;
            case 2:
                text = "Earth";
                break;
            case 3:
                text = "Fire";
                break;
            case 4:
                text = "Air";
                break;
            case 5:
                text = "Thunder";
                break
        }

        //You can change the x and y coordinates
        //if the word doesn't fit the menu
        TextRenderer.drawKeywordText(x - 30, y, text, -1, color, font);
    }
```

As you can see in the drawUnitParameter function, we can fetch the current value of the stat by using statusEntry.param. Depending on which value it returns, we use a different string of text, and then we use TextRenderer.drawKeywordText to draw the text in the menu screen.



New Unit Lv 1 Ex 0
Lord HP 20 / 20

Short Sword 30

Str	7	Mag	0
Skl	4	Spd	3
Lck	0	Def	3
Res	0	Mov	6
New	4	Aff	Earth

Atk	17
Hit	87
Crt	4
Avo	6
Rng	1

-Supporter-

Finally, we need to override the function isParameterDisplayable. This function returns true by default, but there are certain scenarios where you would not want to show the stat. For example, the max HP doesn't show in the same window as the other stats, but instead is rendered in a different place along the current HP. But the max HP stat is displayed during a level up screen. In this case, we want to do the opposite: show it only in the menu screen, and not display it during the level up screen. As a reference, this is the code for the max HP stat.

```
UnitParameter.MHP = defineObject(BaseUnitParameter,
{
    //...

    isParameterDisplayable: function(unitStatusType) {
        // Display if it's not the unit menu.
        return unitStatusType !== UnitStatusType.UNITMENU;
    }
}
);
```

As you can see, the parameter of the function tells you which type of situation the game is. If the value is UnitStatusType.UNITMENU, it means that the game is in the unit menu scree. In any other case, the value is UnitStatusType.NORMAL.
So in the case of the affinity state, we want it to be displayable only if the value is UnitStatusType.UNITMENU.

```
    isParameterDisplayable: function(unitStatusType) {
        return unitStatusType == UnitStatusType.UNITMENU;
    }
```

Other details

While the guide already covers all the basic things you need to create a stat, there are some details that were left unexplained, mainly because their relevance is minor.

getMinValue

This function determines the min value possible for the stat. It is only used by the max HP stat, which sets it as 1, because it doesn't make sense for a unit to have a max HP of 0. If you want to add a min value, you can do the same as the getMaxValue. Assign a new parameter to the class, called "newStatMin", and use it as the value to return.

```
    getMinValue: function(unit) {
        var value = 0; //Default value
        var unitClass = unit.getClass();
        if (unitClass.custom.newStatMin != null) {
            value = unitClass.custom.newStatMin;
        }

        return value;
    }
```

Using Parameter Bonus Skill

As it was already explained before, each data that has the Parameter Bonus option can be used by the getParameterBonus function to give a bonus to the stat of the unit. These types of data are classes, weapons, items and states. But there is another option that is a bit hidden: skills of the Parameter Bonus type. For that type of skill, you do the same as the other types of data: instead of using the Parameter Bonus window (called Skill Effects for the skills) you add the custom parameter associated with the stat, which in this case is newStateBonus.

Sample code

This guide comes with a file called new-stats.js that comes with all the code explained above already working, so you can test it yourself and use it as reference to create your own stats.