Heroes of Newerth

Sol's Theatre

A map making guide

Introduction

Welcome to Sol's Handbook, your interactive guide for Heroes of Newerth's (HoN) map making editor.

Most of you are probably aware that HoN is powered by a custom K2 Engine that helps deliver enhanced graphics while maintaining low system specs. What you may not know is that HoN's K2 Engine is capable of interpreting two syntaxes: Lua and XML. Sol's Handbook will cover the basic steps required to create a map in HoN's map editor using XML

Before you get your hands dirty with HoN's map editor, you'll need a few things:

Heroes of Newerth

Once you have the latest version of HoN installed on your system, we recommended making a copy. Moving forward you'll want to make ALL modifications in your replicated folder so you don't break anything in your regular copy.

A Map

This tutorial will teach you how to craft a map from the ground up using XML; however, to save time and hassle, we recommend finding a pre-crafted map to use as a foundation.

A text editor for code, markup and prose

A good text editor will make a world of difference when sorting through lines of code. There are plenty of free options out there. We recommend Notepad++ and Sublime Text 3.

A Zip Editor for .s2z files

Odds are that you're using WinRAR for your zip needs; unfortunately, due to recent updates the newest version of WinRAR will not read LZ77/78 compression algorithms. S**aving changes to any .s2z file with WinRAR versions above 4.0 will corrupt your HoN folders.**

As a solution, simply uninstall your current copy of WinRAR and download an older version.

Getting Started

Step 1: Create a copy of your Heroes of Newerth client

The first step involves creating a copy of your HoN client. Simply Copy and paste your file and rename it. Moving forward you'll want to make ALL modifications in your replicated folder so you don't break anything in your standard copy.

Step 2: Enable File Extensions

Before diving in to HoN's file database, we recommend enabling file extensions. Enabling file extensions will show you what file types you are working with.

Step 3: Locate Resources

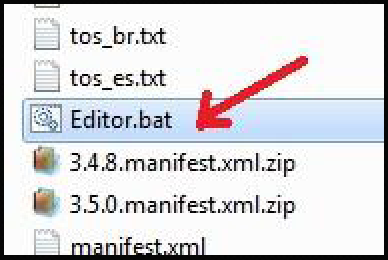
Next, you'll want to locate HoN's Game folder:

Start > Computer > Local Disk > Program Files (x86) > Heroes of Newerth > Game.

Upon opening HoN's game folder, you'll see a .s2z file called resroucecs0.s2z. With the exception of textures, all of HoN's resources are housed in this folder. **To view its contents, you'll need to open it with a zip editor (see prerequisites).**

Opening HoN's Map Editor

Before beginning the map creation process, you'll need to open HoN's map editor. To do so, You'll need to access a file called **editor.bat** which is located in the main **Heroes of Newerth** folder.



tos_br.txt
tos_es.txt
Editor.bat
3.4.8.manifest.xml.zip
3.5.0.manifest.xml.zip
manifest.xml

Controls

WSAD – Keys used for navigation
Mouse Wheel – Hold to alter your map perspective; you can also scroll to zoom in and out

Map Size
To manipulate the size of our map click **File** at the top left and **New Map**

There are 3 different options for setting up the map that work with each other.

Map Size: Multiplies the value of map scale/texture scale, it is highly recommended to be left as 9

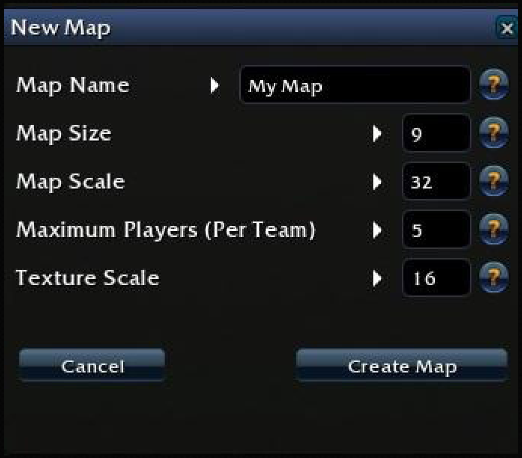
Map Scale: The size of the map, the default is 32 (caldavar's size)
Maximum 64, the bounds radius will start bugging out if made any bigger than this.

Texture Scale: Size of the terrain textures, this needs to be modified when the map scale is changed to retain the same size textures.

Example:

Map Scale = 32
Texture Scale = 16

Map Scale = 16
Texture Scale = 8



New Map

Map Name	My Map
Map Size	9
Map Scale	32
Maximum Players (Per Team)	5
Texture Scale	16

Tools

Deform

-Modify/Raise the terrain, remember that any terrain raised counts as a ramp, you won’t be able to see up it without being at the top.

Paint

Paste textures and colors onto the map

Color Mode changes the color of the terrain using RBG values. White = no color change/default texture. You can right click to remove any color you make (you can use a secondary color, and the secondary one is white by default.

Texture Mode places terrain textures onto the map there are Two Layers.

Layer 1: The first layer is different to the second one, as it has no opacity even with a soft brush.

Layer 2: Unlike the first tool, pastes onto the map based on the brush strength, you cannot paste two layers 2 textures close to each other or they will glitch out.

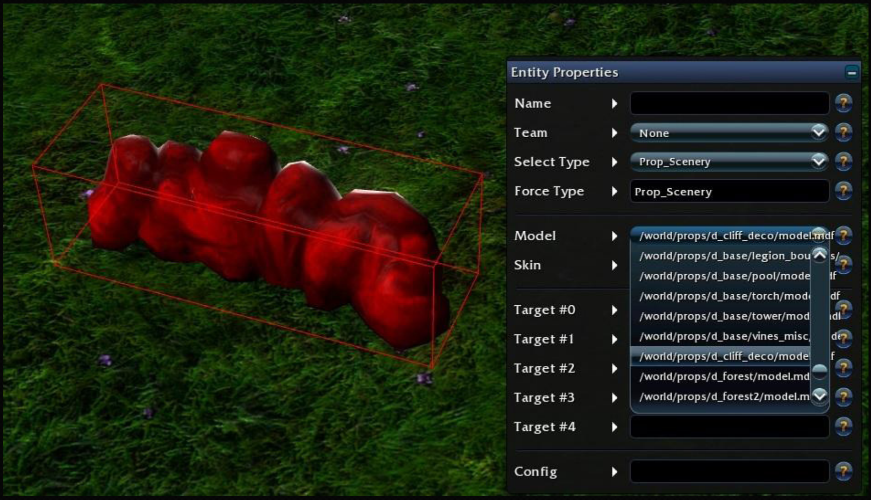
Entities

Use this tool to place entities on your map.

Entities include everything from, projectiles, gadgets, heroes, and neutrals to props, trees, and other objects.

To create an entity, hit the create button at the top and click on the map, by default in the map editor it will make an Entity Type called Prop_Scenery and a teapot will spawn.

A Prop_Scenery is a Client side entity that has no code/interaction with the game, you can change its model by selecting it and clicking the model dropdown box in entity properties


Entity Properties

Name
Team: None
Select Type: Prop_Scenery
Force Type: Prop_Scenery
Model: /world/props/d_cliff_deco/model.mdf
Skin
Target #0
Target #1
Target #2
Target #3
Target #4
Config

```
/world/props/d_base/legion_bou
/world/props/d_base/pool/mode
/world/props/d_base/torch/mod
/world/props/d_base/tower/mod
/world/props/d_base/vines_misc
/world/props/d_cliff_deco/mode
/world/props/d_forest/model.md
/world/props/d_forest2/model.m
```

Changing the model for most other entity types via the entity properties will result in the type you selected using its default model on the .entity file itself in most cases.

You can change the Prop_Scenery to a different entity by clicking the down arrow on the type dropdown list.

Foliage

Decorate the map with grass, flowers, etc.

Select a foliage texture on the right and choose the settings you want to use, remember there are two layers if you want to use two different foliage textures together with different settings inside the same radius.

Sound

Add ambient sounds to the map

Blockers

This tool has two modes, Path Blocker and Visual Blocker.

Path Blocker lets you paste your own collision on the map manually, using the same physics as the collision cliffs have

Visual blocker lets you create an area you can not see past until you have crossed it, even if you're above terrain/ramp.

Cliffs

This tool has three features: **adding cliffs and ramps** and **removing riles.**

Right click increase the height of cliffs; left click to lower.

Click the ramp tool and select the sides of cliffs to add ramps.

The remove tiles function allows you to erase a part of the terrain and make it transparent.

The way cliffs are made is they are actually Entities with the type Prop_Cliff.
They do not have internal collision, instead four things are done automatically when placing them, a visual entity is created, the height of the terrain is raised, a tile is removed, and a blocker is added all at once for each cliff tile.

Water

Creates water

Select a water material you would like to use/brush size and paint away, right click to remove any water you've made.

Note: There is currently a bug with this tool, although the water is actually applying you can't see it, to fix this, click and drag around to paste some water, save map and then load it again and your water will appear.

XML Basics

Sol’s Handbook covers the basics of creating a map in HoN’s editor using XML; for a comprehensive introduction click here.

XML stands for extensible markup language; its main purpose is to carry data.

XML uses a tag based system; tags use letters to define what it is and either ends right away or after the end is specified. The beginning and end of a tag is signified by
A less-than symbol (<) signifies the start of a tag while a greater-than symbol (>) represents its end.

<hero> .. </hero> everything in between the tags are children to hero

With attributes:
<hero name=”Hero_Valkyrie”> .. </hero> everything in between the tags are children to hero
Ending directly
<hero name=”Hero_Valkyrie” /> The tag has no children.

Note how a slash (/) controls a tag’s end. To end a tag directly, append / to the last >o end it at a later point you make an end tag and have the / appended at the start.

Tags control content, attributes control tags. Every XML Document always begins with
<?xml version=”1.0” encoding=”UTF-8”?>
At the top of the page.

Most of the tags used in HoN’s map editor will be introduced later in this tutorial.

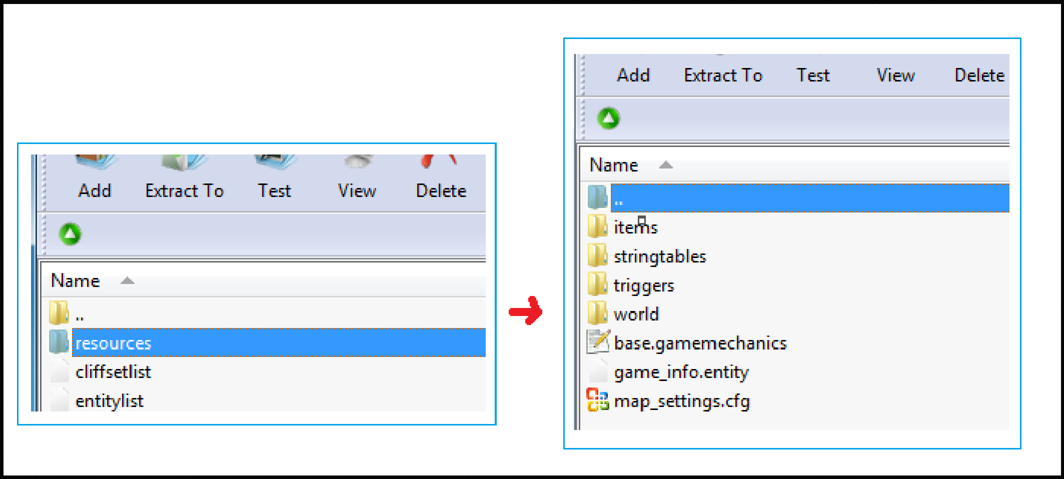
Inside the map

Gadgets

A gadget is an XML file containing the gadget specification.

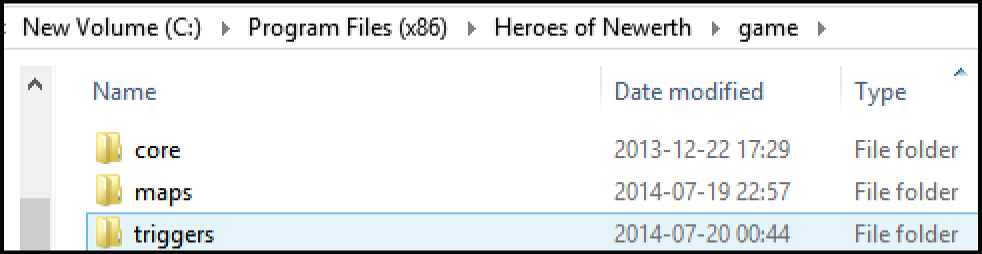
To create a gadget, we'll need a new file. Inside your triggers folder right click, create a new text document open it and save it as “gadget.entity”. Please note that every entity you create has to be saved with a .entity tag.

Incorrect: gadget.entity.txt.
Correct: gadget.entityThe scripts which go into the map go inside the resources folder like this:



If it's not there, create the folder. You just place everything in there later.

To add stuff without needing to restart, inside your game folder create a triggers map like this:


New Volume (C:) ▸ Program Files (x86) ▸ Heroes of Newerth ▸ game ▸

Name	Date modified	Type
core	2013-12-22 17:29	File folder
maps	2014-07-19 22:57	File folder
triggers	2014-07-20 00:44	File folder

Once inside, you can create your first entity.

Every entity must start with the XML Definition first:

<?xml version=”1.0” encoding=”UTF-8”?>

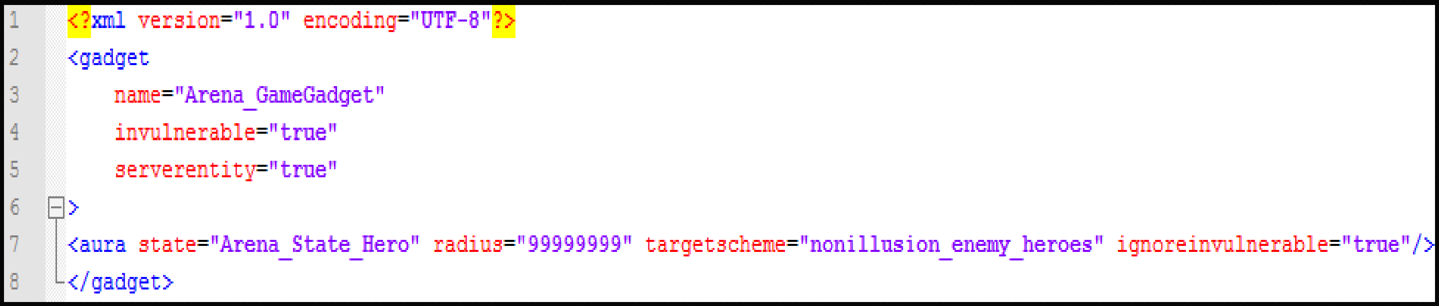
Followed by your entity description which right now should be empty like this:

```
<gadget
	name=”Arena_GameGadget”
>
</gadget>
```

There’s lots of attributes you can add other than name, but this example we’ll use two:

invulnerable=”true” - Prevents it from being damaged
serverentity=”true” - Creates the entity on the server only, it is not a visual element of the user

We also want the gadget to apply a state to all the players. This is what the <aura/> tag is for. We’ll add the tag between the start and end of the entity so the full entity file should look like this now:



```
<?xml version="1.0" encoding="UTF-8"?>
<gadget
    name="Arena_GameGadget"
    invulnerable="true"
    serverentity="true"
>
<aura state="Arena_State_Hero" radius="99999999" targetscheme="nonillusion_enemy_heroes" ignoreinvulnerable="true"/>
</gadget>
```

State supply which state to apply, just like the gadget in entitylist this one needs to exist or you’ll run into problems. Radius supply how long from the gadget it should be giving the state, this one is pretty much global as you may notice. Ignoreinvulnerable is whether to supply invulnerable units with the state too or not, in this case no.

The targetscheme is no easy one to explain. In resources0 you’ll find a file called base. gamemechanics. Search it for “targetscheme” and you’ll find all of them. Their name usually supplies a good description of what they do, however else allow / restrict will explain it for you. nonillusion_ enemy_heroes in this case as our GameGadget is team 0 counts as both the teams and all heroes excluding illusions.

Let’s move on to the state we’ll be supplying units with.

Gadgets

There'll be two gadgets used in this. We'll cover them straight away so you don't have to come back to them later. Both should be fairly obvious in what they do by now but explanations will be supplied.

```
<?xml version="1.0" encoding="UTF-8"?>
<gadget
	name="Arena_GameGadget"
	invulnerable="true"
	serverentity="true"
>
<aura name="Arena_GameGadget_Aura" state="Arena_State_Game" radius="0" targetscheme="self" ignoreinvulnerable="true"/>
<aura name="Arena_Dead_Aura" state="Arena_State_Hero_Dead" radius="9999999" targetscheme="all_dead_heroes" />
</gadget>
```

The aura only hits the gadget itself and since it is invulnerable this state will be on this gadget until the game ends. The state on it will be Arena_State_Game which will come further down.
There's a state on all heroes which are dead to prevent them from respawning.

```
<gadget
	name="Arena_ScoreTracker"
	invulnerable="true"
	serverentity="true"
>
</gadget>
```

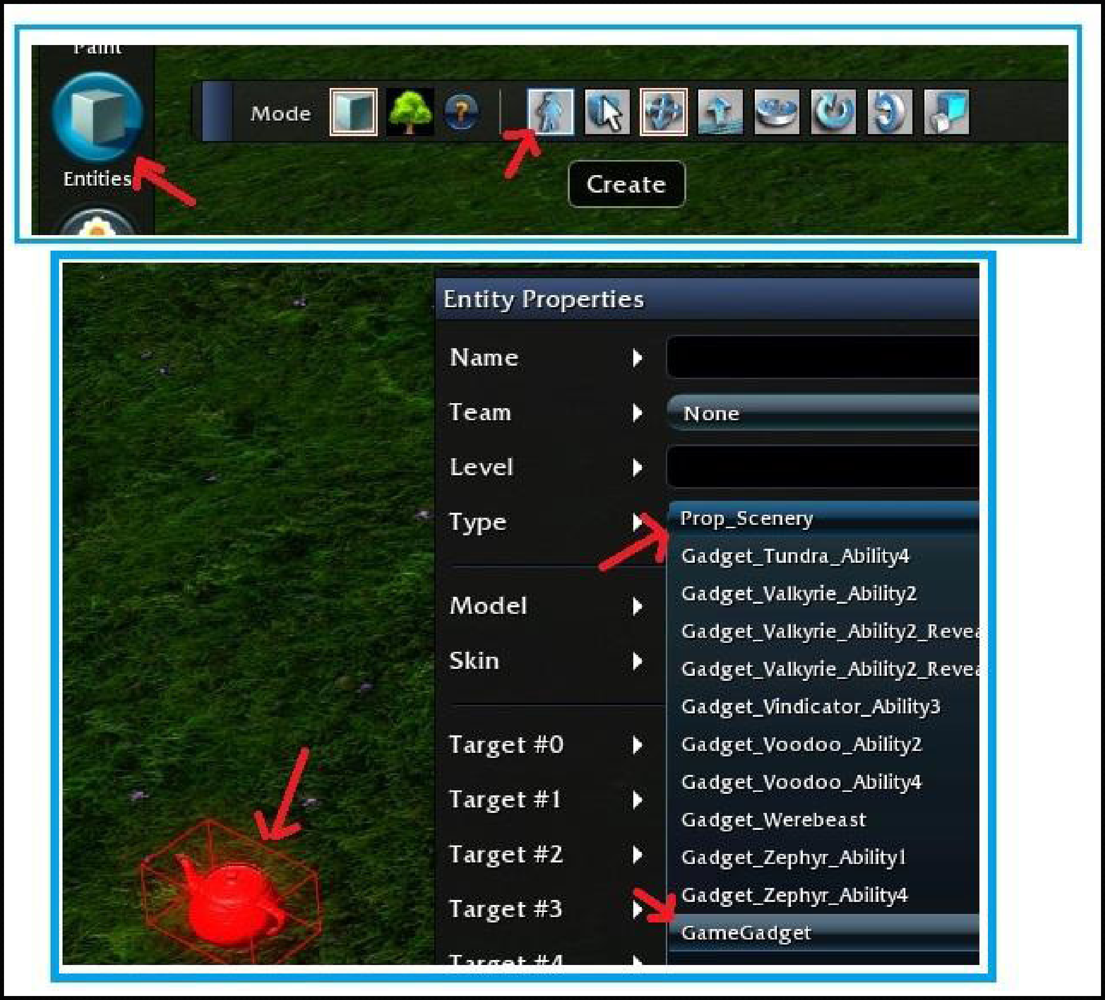
And that's self explanatory, it may seem pointless and it partly is but it's a very good tool for us to use.

The Entity list

The Entity list contains entries for entities to be automatically spawned when your map initializes. Adding to it is the first step to script custom content. Normally, adding a gadget that maintains the game is very good practice.

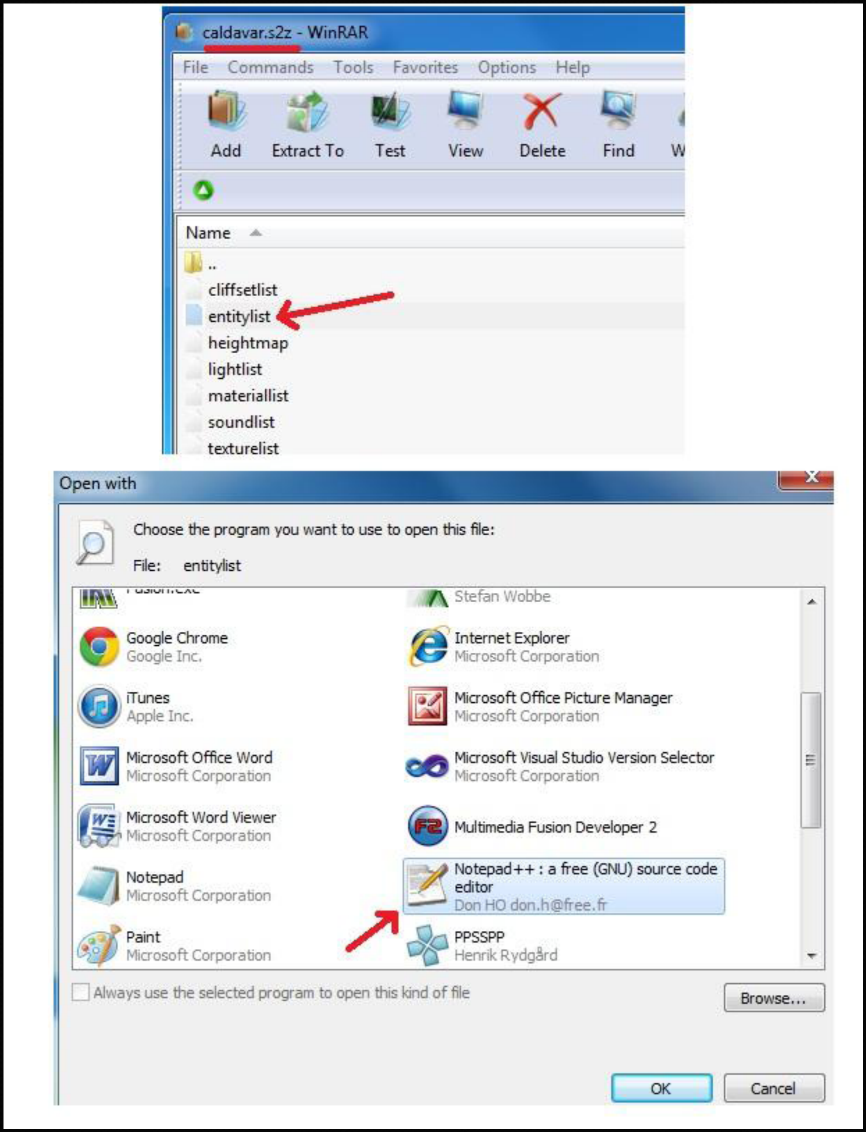
There are two ways to add entities to the map, manually adding the entity list file inside the map, or placing it inside the Map Editor.

The first way is to place it inside the map editor, open it up and select Entities Create, and place a teapot on the ground, and change its type to GameGadget.



(Note: You will only see custom entities in the type dropdown box if it is extracted in game folder, if the files are inside the map only it won't appear even if you load the map in editor with them inside it.)

The second way is to manually change the Entity List.
Open the map you wish to edit, and open the file entity list in notepad++


caldavar.s2z - WinRAR
File Commands Tools Favorites Options Help
Add Extract To Test View Delete Find
Name
..
cliffsetlist
entitylist
heightmap
lightlist
materiallist
soundlist
texturelist
Open with
Choose the program you want to use to open this file:
File: entitylist
Google Chrome
Internet Explorer
iTunes
Microsoft Office Picture Manager
Microsoft Office Word
Microsoft Visual Studio Version Selector
Microsoft Word Viewer
Multimedia Fusion Developer 2
Notepad
Notepad++ : a free (GNU) source code editor
Paint
PPSSPP
Always use the selected program to open this kind of file
Browse...
OK
Cancel

An entity list contains a long list of entries to be spawned and each one look something like this.

<entity angles=”0.0000 0.0000 0.0000” index=”0” model=”/shared/models/invis.mdf” name=”GameGadget” position=”8192 8192 0” scale=”1.0000” seed=”0” skin=”default” team=”0” type=”Arena_GameGadget”></entity>For this example we won’t worry about angles, name, seed or skin.
Index needs to be a unique ID for you gadget. Nothing else in the entitylist can have the same index. Just put it at the bottom and choose whatever index is one more than the previous last.

The model defines how this will look in game, this gadget will be invisible to the players and thus it doesn’t have a visible model. The invis.mdf default model is an empty model.

The scale defines the size, for props this is how big it LOOKS, for other type how big it acts. On gadgets like these, the look is defined inside the gadget as modelscale. More on that later.

Position chooses where you’ll place your gadget when the map starts. The position above was chosen to be the exact center of caldavar which is 16384 x 16384 (2^9 x 32) 9 being size, 32 being scale of the map. 0 is the height which won’t matter for <aura/> effects which we plan to use this for.

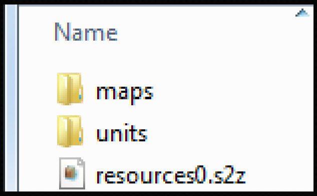
Team specify who it belongs to, 0 being nobody (the empty team), 1 legion and 2 hellbourne.

The type is what Entity Type to place. You need to make sure there’s a script, and only one with the name of whatever’s in the Type. Without further ado, let’s get down to the gadget script and see what we can do about that.

Units

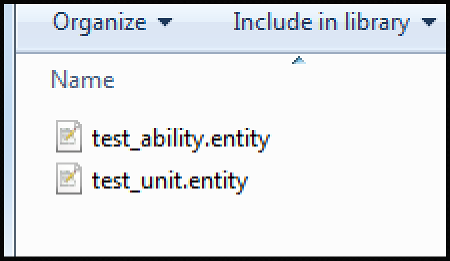
For this part of the tutorial we're going to create a very simple test Unit.

Create a folder called "units" in the game folder.



Create two new files (fastest way: Create New > Text Document)

Rename the two files: test_unit.entity / test_ability.entity



Let's make a simple unit first, open up the test_unit.entity you just created.

The K2 engine has a few different types for different units.

<hero> / <neutral> / <creep> / <pet> / <critter>

For this example we'll use a very simple unit type: <pet>

```
<?xml version="1.0" encoding="UTF-8"?>
<pet


>



</pet>
```

Unit type’s need a-lot of attributes to function correctly, it’s quicker to have a template for new heroes/units, or even just copy paste them from another hero and edit them.

You will need more than these for a proper unit entity, for a full list of attributes for units go here!
<hero> / <pet> / <gadget> / <creep>

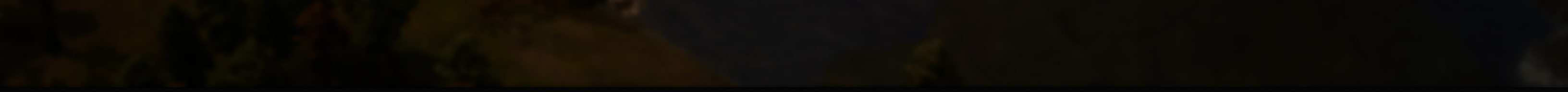
Or open up one of the heroes from /heroes/ inside resources0.s2z as a guide.

Here are the absolute essentials for a Unit to function correctly:

name=””	-	Type of Unit
icon=””	-	Icon
model=””	-	Model
modelscale=””	-	Size of model
effectscale=””	-	Size of effects played on the unit
boundsradius=””	-	Size of bounds (for collision/selecting unit)
boundsheight=””	-	Size of height bounds (for selecting unit)
selectionradius=””	-	Size of green selection around unit
movespeed=””	-	Base movement speed
turnrate=””	-	How fast you can rotate when turning
maxhealth=””	-	Base health
maxmana=””	-	Base mana
armor=””	-	Base armor
magicarmor=””	-	Base magic armor
inventory0=””	-	Ability - Q
inventory1=””	-	Ability - W
inventory2=””	-	Ability - E
inventory3=””	-	Ability - R
inventory4=””	-	Ability - O (Attribute)
inventory5=””	-	Ability - D
inventory6=””	-	Ability - F
inventory7=””	-	Ability - G
inventory8=””	-	Ability - N (Taunt)
inventory9=””	-	Ability - U
inventory10=””	-	Ability - K
inventory11=””	-	Ability - J
attackduration=””	-	total time for the entire attack (ms)
attackactiontime=””	-	time of duration on attack damage impact (ms)
attackcooldown=””	-	cooldown of attack after duration ends (ms)
attackdamagemin=””	-	minimum damage
attackdamagemax=””	-	maximum damage
attackrange=””	-	attack range
attacktype=”melee”	-	set if ranged (projectile) or melee attack

For this example use these settings, this uses an M&M halloween candy model/icon as the unit from resources0.s2z with some basic settings/sizes and our upcoming new ability “Ability_Halloween1” set to Q

```
<?xml version=”1.0” encoding=”UTF-8”?>
<pet
    name=”Pet_Halloween”
    icon=”/tools/m_c.tga”
    model=”/tools/m.mdf”

    modelscale=”1.0”
    boundsradius=”25”
    boundsheight=”115”
    selectionradius=”35”

    movespeed=”350”
    turnrate=”450”

    maxhealth=”500”
    maxmana=”250”

    armor=”1.0”
    magicarmor=”1.0”

    inventory0=”Ability_Halloween1”
    inventory1=””
    inventory2=””
    inventory3=””
    inventory4=””
    inventory5=””
    inventory6=””
    inventory7=””
    inventory8=””
    inventory9=””
    inventory10=””
    inventory11=””

    attackduration=”1000”
    attackactiontime=”500”
    attackcooldown=”1250”
    attackdamagemin=”45”
    attackdamagemax=”50”
    attackrange=”128”
    attacktype=”melee”

>

</pet>
```



Abilities

Now to script the ability we added to the Unit earlier.

As before with the hero there are a-lot more attributes then these, it is highly recommended to study how other abilities are made in resources0.s2z
(example: /heroes/dwarf_magi/ability_01/ability.entity to see Blacksmith’s Q)

Or for a full list go Here!

Here are the essential attributes for the basic ability we’re going to make:

name=””	-	Type of Ability Entity
icon=””	-	Icon
casttime=””	-	Cast time (ms)
baselevel=””	-	Level of ability by default
maxlevel=””	-	Maximum level
requiredlevel=””	-	Required hero level to level up ability
actiontype=””	-	Type of cursor usage List Of Action Types
casteffecttype=””	-	Type of element
targetscheme=””	-	Who is targeted by ability List Of Target Schemes
manacost=””	-	Mana cost
cooldowntime=””	-	Cooldown time (ms)
range=””	-	Range of Ability

If you want to make it so the mana cost / range / option to level up changes on the heroes level, add a comma for each level, works slightly different for some attributes.

Example: requiredlevel=”1,3,5,7”
Makes it so you need your <Hero> to be level 3 to level the <Ability> up to 2, and <Hero> level 5 to level the <Ability> up to 3.
Bonus: Ultimates are normal abilities with requiredlevel=”6, 11, 16” and maxlevel=”3”

Example 2: manacost=”100,135,150,175”
When the <Ability> is level 1 it costs 100 mana, When the <Ability> is level 2 it costs 135 mana, etc.

For this example I’m going to be demonstrating how to make a simple target ability that does damage, stuns, and plays an effect.

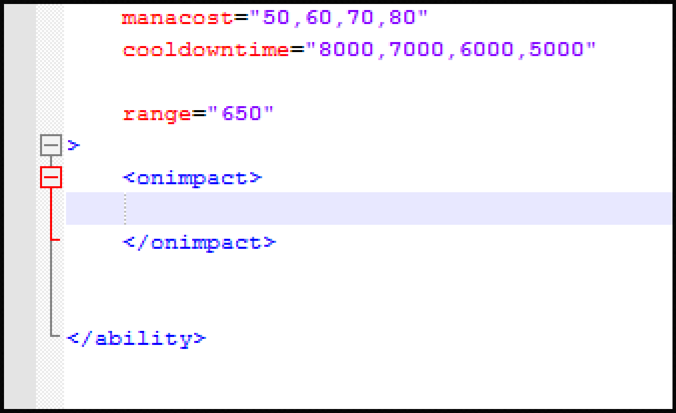
We will name the ability to what we named it in the Unit “Ability_Halloween1”

Use these settings;

```
<?xml version=”1.0” encoding=”UTF-8”?>
<ability
   name=”Ability_Halloween1”
   icon=”/tools/m_c_red.tga”
   casttime=”250”

   baselevel=”1”
   maxlevel=”4”
   requiredlevel=”1,2,3,4”

   actiontype=”target_entity”
   casteffecttype=”Magic”
   targetscheme=”enemy_units”

   manacost=”50,60,70,80”
   cooldowntime=”8000,7000,6000,5000”

   range=”650”
>

</ability>
```

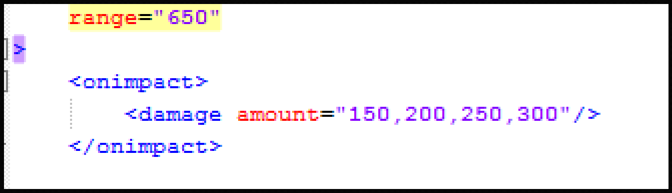
The ability is setup but hasn’t got any scripting for what it does if it hits someone yet.

Add an event underneath between <ability> and </ability> named <onimpact>



```
    manacost="50,60,70,80"
    cooldowntime="8000,7000,6000,5000"

    range="650"
>
    <onimpact>

    </onimpact>


</ability>
```

We want to make the ability do damage when the spell impacts the enemy.
Impact means when the projectile or the ability itself hits a target, this can be after a delay or instantly when clicked.
Add the action to the onimpact event called <damage> with an amount=”” attribute, add 4 different levels of damage like this:



```
    range="650"
>

    <onimpact>
        <damage amount="150,200,250,300"/>
    </onimpact>
```

The spell should will do now do damage, let’s add a few more things.

```
<applystate name=”” duration=”” />
```

This applies a state to the target (in this case the one that got impacted)

```
<playeffect effect=”” source=”” target=”” />
```

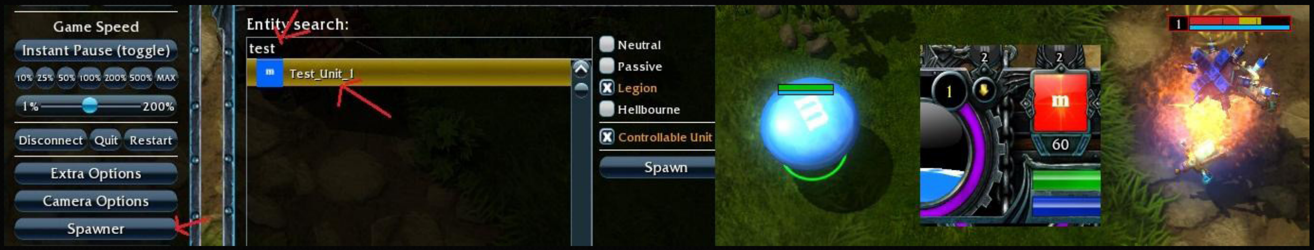
This plays an .effect file on the target.

.effect files are what makes all the spell effects/particles and other cool graphics you see in the game, they are made in XML and are open source.
Let’s add a Stun State, and play blacksmith’s Q impact graphic.

```
<onimpact>
            <applystate name=”State_Stunned” duration=”500,1000,1500,2000”/>
                       <playeffect effect=”/heroes/dwarf_magi/ability_01/effects/impact.effect” source=”target_entity”/>
                        <damage effecttype=”Magic” amount=”150,200,250,300”/>
   </onimpact>
```

All done! Your Unit is ready to be tested in game!

Enter a practice game and open the spawner in Test++ and search for test unit and spawn it. Spawn an enemy unit also of some sort (like some other hero) and see if everything is working!



Figuring out the actions you can use like <areaofeffect/> and so on can be the most tricky part. Struggling with the actions is something every developer does until they start using a wide range of them efficiently and often. A list of events and actions can be found here.

Creating a Round Based Scoring System

Beware, it's about to get a little techy. I assume you've familiarized yourself a little bit with how entities work by now and at least read the above sections. First off, repeat the steps from section 1 by creating a GameGadget and adding it to the map. Skip the state that example used, we'll make a different state for the sole purpose of this. Do not that the conditions for a round will be the following:

If only one team is alive, they win that round.
If noone is alive it is a tie. This can occur if two people from different teams die on one frame.
If two people are alive and they are from different teams, the round continues until only one team is alive.

The scoring system will then be counted like this:
If a team wins, they gain one point.
If it is a tie, no one gains any point.

When a round ends regardless of outcome, everyone is respawned automatically. You cannot spawn if you have died during a round.

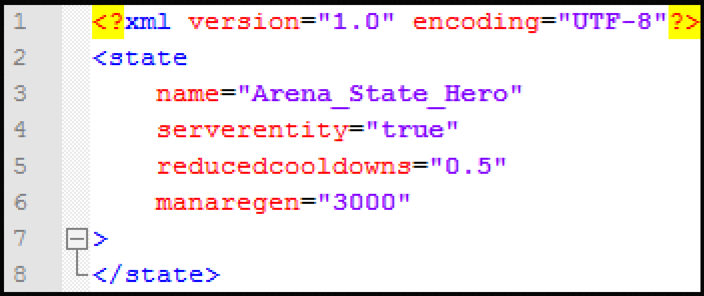
States

States can do lots of things, for this example we'll again keep it straight and simple. I just want to show you how two simple files create something amazing for you. Create a new file again, call it state.entity. We'll be making the name the same as the state above, add a serverentity="true" attribute again and instead of <gadget> it's going to be <state>. It should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<state
	name="Arena_State_Hero"
	serverentity="true"
>
</state>
```

We'll be adding two more properties to make this state interesting.
reducedcooldowns="1"
manaregen="3000"

If you can guess what these do, good job. The final state:



```
<?xml version="1.0" encoding="UTF-8"?>
<state
    name="Arena_State_Hero"
    serverentity="true"
    reducedcooldowns="0.5"
    manaregen="3000"
>
</state>
```

And that's it! Saved the entitylist and these two states you can now go ingame and play. If you took the advice to create caldavar_test.s2z go to hosted games, create game choose "practice" and for map there should be an entry called map_caldavar_test

So you've got your first taste of states but these next steps are a little tough, as states can be long and contain complex logic.

Let's continue with Alive / Dead states:

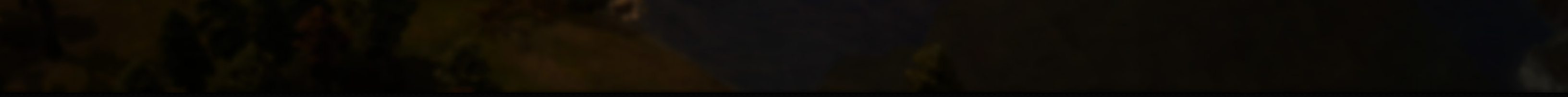
```
<?xml version="1.0" encoding="UTF-8"?>
<state
	name="Arena_State_Hero_Dead"
	serverentity="true"
>
	<oninflict>
		<setrespawntime a="0" />
		<preventrespawn value="true" entity="target_entity" />
	</oninflict>
</state>
```

This command prevents anyone alive from respawning. Target_entity is the hero, value="true" means to prevent, setrespawntime a="0" has a target already which is the player.

Respawn state is pretty straight forward too:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<state
        name=”Arena_State_Respawn”
        serverentity=”true”
>
        <oninflict>
                <expirestate name=”Arena_State_Hero_Dead” target=”target_entity” />
                <preventrespawn value=”false” entity=”target_entity” />
                <setrespawntime a=”0” />

                <!-- Give gold or do whatever you wanna do to heroes between rounds :) -->
        </oninflict>
</state>
```

The hard part is the game tracking state. There’s comments for you but that’s as much as can be given. It’s too long for this document, here’s a link to it: http://pastebin.com/expT7193



© 2014.