# Script - Tutorial 1

HPL2 uses a script language called AngelScript, it is a C/C++ based syntax and a brief overview of the syntax can be found here.

## I do not know how to script?!

Scripting in HPL2 is about as easy as it gets, without adding an extra help system (for example Blizzard has some very nifty tools in their editors to help with scripting). But even so it will be a mess and you will be lost with no previous experience. What you need is basically an introduction course to C/C++ or a script language based on that syntax, for example Javascript.

In my own experience, as a non-educated fellow in the world of scripting, I think you are best of taking a look at Javascript before you try to begin scripting in HPL2. Why? Because it is easy to find Javascript tutorials and information online and all you need is a text editor and your web browser. A place to start is for example w3schools introduction to Javascript. Mainly chapters JS Comments to JS For...In touch the general subjects that are common to all syntax's.

A short version is that all that you do in HPL2 is to use functions, they are commands that you give properties and then all the magic is taken care of by the game. Everything here are functions. You then decide how and when the functions are to be executed by using the common programing specific parts such as *statements*, *variables* and *comparisons* in combination with the objects you place using the level editor.

### Example for those with no prior experience

Script is simplified and **will not actually work ingame AT ALL**, syntax is incorrect and only written as it is to give a short snippet of script code.

```
AddEntityCollideCallback("Player","AreaHelp","CollidePlayerWithAreaHelp");

void CollidePlayerWithAreaHelp()
{
    if(HasItem("cellar_key") == true)
    {
        SetMessage("Use your inventory to select key and use on door.", 10);
    }
    else
    {
        SetMessage("You will need a key to open the door.", 5);
    }

}
```

Imagine that, using the level editor, we have created a simple room. In this room we have a locked door and in front of that door we have added an area called “AreaHelp”. What we now have done in

the script is that we have added a callback (a trigger if you so like), this is the line with AddEntityCollideCallback. It says that when the Player enters AreaHelp it should trigger the function CollidePlayerWithAreaHelp.

- When the player enters this area the function (begins with void CollidePlayerWithAreaHelp() and the rest of the script) is triggered and this occurs:

1. It asks, does the player have the item “cellar_key” in his inventory?
    1. If this is true, we will then show a message on the screen for 10 seconds telling him what to do with the key.
    2. If this is not true, we show a message about where to find the key for 5 seconds instead.

This is all there is to it. Sure things will be more complicated and much longer, but the whole idea is always the same. When something occurs that you are looking for, you ask if something is the way you want it to, depending on what the answer is, you do different things. The tricky part will be to learn all the specific bits that you can do, what the game allows for, how to write something that affects many objects, but without doing it specifically for each (this is where variables comes in) and that sort of thing (always being logical is the key, but that is hard for a human (me)!). Imaging it as being a Swede and learning English:

You learn that “Flytta” can be “Move” in English, that you can *Move* an object, say a *Rock*, and that the *Rock* can be *Moved* at different *Speeds*. In HPL2 this would be a function you can call in the script, for example (again this does not really work):

```
MoveObject("Rock", 10);
```

How this actually makes the rock move is not important, that is all taken care of by the game, but you have specified that you want to move a specific Rock and that is should be moved at a speed of 10 (this could be 10 m/s, but this is something you need to look up and learn and not something you specify). The only HPL2 specific thing here would be *MoveObject*, which is the name of our function (word) and the *(“”,);* are the grammar (the syntax for C/C++ like scripts) and *Rock* is a name of an object in the level editor and *10* is a specified value.

Good luck and remember to read and learn ALL script functions available, or just bookmark that page. ;)

# Scripting with HPL2

A script file is a plain text file, with the same name as your map file. For example: my_test_level.map should have a my_test_level.hps text file in the same location. A script file can be edited using good old Notepad, if you do a lot of scripting it might be a good thing to read about Notepad++.

For the below to work you need a very simple test map. All you need is to make a plane to stand on, a PlayerStartArea to have a place the player will start at and a light so that you can see. In the last example you will need to add a Script Area too and naming it “AreaTest01”. See this tutorial for map creation.

1. The script file is quite straightforward, begin by copying and pasting the first example below

into your text file. Then start the game with your map and you should be able to activate a lantern, if you look in the inventory you should also have 10 tinderboxes. If you do not, make sure that you have setup a development environment, failing to do this will make all debug stuff not work.
2. In the second example there are two new things: A AddTimer in OnStart and a new function.
3. In the third example, an AddEntityCollideCallback was added in OnStart and a new function for it. You also add an area to your level and call it “AreaTest01”, placing it close to the PlayerStartArea.

## A default .hps file

```
//////////////////////////
// Run first time starting map
void OnStart()
{
    //Add the Lantern and 10 Tinderboxes when in Debug mode, always good to have light!
    if(ScriptDebugOn())
    {
        GiveItemFromFile("lantern", "lantern.ent");

        for(int i=;i<10;i++) GiveItemFromFile("tinderbox_"+i, "tinderbox.ent");
    }
}

//////////////////////////
// Run when entering map
void OnEnter()
{

}

//////////////////////////
// Run when leaving map
void OnLeave()
{

}
```

## Adding a timer function

In OnStart we now add:

```
AddTimer("tut01", 5, "TimerTutorial01");
```

Then anywhere in the script file, as long as it is NOT inside OnStart, OnBegin or OnLeave, we add:

```
void TimerTutorial01(string &in asTimer)
{
    SetMessage("Inventory", "CombineLevel12Error", );
}
```

Making the whole script file look like this (with comments added to explain what is going on):

```
//The Timer function that triggers after 5 second from first time map loads.
void TimerTutorial01(string &in asTimer)
{
    //Displays a message on screen. Category and Entry are the two strings,
as read from the file
    //redist/config/lang_main/english.lang (if using English).
    //0 is the time it will be shown (0 means it auto-calculates the time
based on length of message).
    SetMessage("Inventory", "CombineLevel12Error", );
}

////////////////////////////
// Run first time starting map
void OnStart()
{
    //A timer named tut01, that triggers after 5 second the function
TimerTutorial01
    AddTimer("tut01", 5, "TimerTutorial01");

    //Add the Lantern and 10 Tinderboxes when in Debug mode, always good to
have light!
    if(ScriptDebugOn())
    {
        GiveItemFromFile("lantern", "lantern.ent");

        for(int i=;i<10;i++) GiveItemFromFile("tinderbox_"+i,
"tinderbox.ent");
    }
}

////////////////////////////
// Run when entering map
void OnEnter()
{

}

////////////////////////////
// Run when leaving map
void OnLeave()
{
```

```
}
```

## Adding a collide function

In OnStart we now add:

```
AddEntityCollideCallback("Player", "AreaTest01", "CollideAreaTest01", true,
1);
```

Then anywhere in the script file, as long as it is NOT inside OnStart, OnBegin, OnLeave or TimerTutorial01, we add:

```
void CollideAreaTest01(string &in asParent, string &in asChild, int alState)
{
    RemoveTimer("tut01");

    AddDebugMessage("Entered area!", false);
}
```

Making the whole script file look like this (with comments added to explain what is going on):

```
//The Timer function that triggers after 5 second from first time map loads.
void TimerTutorial01(string &in asTimer)
{
    //Displays a message on screen. Category and Entry are the two strings,
as read from the file
    //redist/config/lang_main/english.lang (if using English).
    //0 is the time it will be shown (0 means it auto-calculates the time
based on length of message).
    SetMessage("Inventory", "CombineLevel12Error", );
}

//The collision function that triggers when player enters area AreaTest01
void CollideAreaTest01(string &in asParent, string &in asChild, int alState)
{
    //Removing the timer tut01, so if player trigger this function before 5
seconds has passed since map
    //was loaded, the function TimerTutorial01 will not occur.
    RemoveTimer("tut01");

    //A message that is only displayed when in debug mode.
    AddDebugMessage("Entered area!", false);
}

//////////////////////////
// Run first time starting map
void OnStart()
{
    //A timer named tut01, that triggers after 5 second the function
```

```
TimerTutorial01
    AddTimer("tut01", 5, "TimerTutorial01");

    //If the player enters an area named "AreaTest01", the function
CollideAreaTest01 will trigger.
    //true, means this will only happen 1 time, set to false to happen every
time the player collides
    //with the area.
    //1 means only when player enter area, -1 would be when leaving area and
0 would be both.
    AddEntityCollideCallback("Player", "AreaTest01", "CollideAreaTest01",
true, 1);

    //Add the Lantern and 10 Tinderboxes when in Debug mode, always good to
have light!
    if(ScriptDebugOn())
    {
        GiveItemFromFile("lantern", "lantern.ent");

        for(int i=;i<10;i++) GiveItemFromFile("tinderbox_"+i,
"tinderbox.ent");
    }
}

////////////////////////////
// Run when entering map
void OnEnter()
{

}

////////////////////////////
// Run when leaving map
void OnLeave()
{

}
```