# Dynamically Attaching Scripts

creationkit.com/index.php

There are times that we may want to dynamically attach scripts to objects and actors. This could be because we don't want to alter the vanilla records, or because we think it's going to be too much work to edit every single vanilla record, or because we want our mod to work on objects added by other mods.

There are many ways to do this, and this tutorial will only cover two of them. The first method is to attach magic effect scripts to actors, and the second is to attach reference alias scripts to objects.

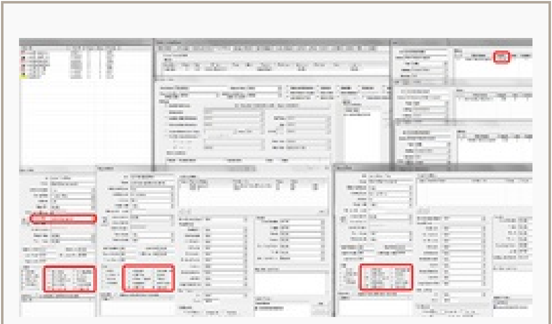
## Contents

## Attaching Scripts to Actors

We're going to attach scripts to Actors by making use of Skyrim's Magic Effect archetypes. We're going to give the player an ability (Cloak), and that Cloak is going to give--via a scripted, aimed concentration spell--nearby Actors their own abilities containing scripted magic effects. For the purposes of this tutorial, we will be placing scripts on Actors which will track the damage done to them by the player, and display the damage as a notification.

### Abstract

This method consists of:

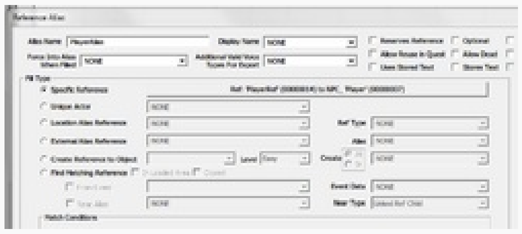

Overview

1. A **Quest**, **`mymodQuest`**, which fills an **Alias** on the player.
2. A Cloak Ability, **`CloakAbility`**, and...
    - The Effect, **`CloakEffect`**, which cloaks an area around the player.
3. A Concentrated, Aimed Spell, **`ApplyingSpell`** and...
    - The Effect, **`ApplyingEffect`**, which will determine whether or not to add a script to the Actor based on various conditions.
        - The Script, **`ApplyingScript`**, which is responsible for adding the 'MonitorAbility' spell to the Actor, and if necessary carrying out any additional conditional logic.
4. An Ability, **`MonitorAbility`**, which is added to the Actor by the above script and...
    - The Effect, **`MonitorEffect`**, which contains...
        - The Script **`MonitorScript`**, which runs on the Actor and will monitor the damage done by the player.

Additionally we will cover the various **Condition Functions** which can help ensure the script does not get applied to Actors unnecessarily, and also show how condition functions can reduce the frequency of unnecessary cloaking checks. Lastly, we will cover how to deal with issues like the **'Brawl Bug'**.

For the sake of brevity and readibility, the prefix "mymod" has been removed from the Spell and Magic Effect names when compared to the tutorial images. It is recommended that you prefix all your forms with a unique identifier.

## Set Up a Reference Alias


Reference Alias

In the **Object Window**, navigate to **Quest** under the Character category. Create a new quest by right-clicking on the list and choosing 'New'. For this tutorial we will be naming the quest **`mymodQuest`**. Make sure that **Start Game Enabled** in the **Quest Data** tab is checked. Click on the **Quest Alias** tab, then right-click in the empty list and choose 'New Reference Alias'. We will name this alias **`PlayerAlias`**. In the Reference Alias window, under Fill Type, select the **Specific Reference** option and click the 'Select Forced Reference' button. Under Cell choose `(any)`, and Ref should default to `PlayerRef (Player)`. If not, select it from the dropdown menu.

## Create the Cloak Ability

First we will set up the Cloak effect which we will put onto the Cloak Ability. In the **Object Window**, navigate to **Magic Effects**, under the Magic category. Create a new Magic Effect by right-clicking on the list and choosing 'New'. Change the Effect Archetype to `Cloak`, set Casting Type to `Constant Effect`, and Delivery to `Self`.

Important Flags

**Hide in UI** - If you want this ability hidden from the player.

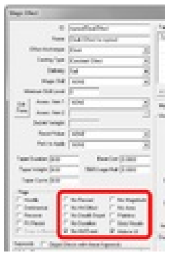
**No Hit Event** - We *must* check this flag as one of the steps to avoid the **'Brawl Bug'**.

We'll give this the ID **`CloakEffect`** and a descriptive Name, and then click OK to accept the changes.

Next, we create the Cloak ability that will be placed on the player. In the **Object Window**, navigate to **Spells** under the Magic category. Create a new Spell by right-clicking on the list and choosing 'New'. We'll give this the ID **`CloakAbility`** and a descriptive Name. Change the Effect Archetype to `Ability`, set Casting Type to `Constant Effect` and Delivery to `Self`. In the Effects list, we will add **`CloakEffect`**. Set Magnitude to `192.0` and leave the Conditions list blank for now, but take note as we will fill this in later as part of working around the **'Brawl Bug'**. Click OK to accept the changes.

The magnitude of a Cloak archetype effect is the range of the cloak in feet. So a magnitude of `192.0` is 192 feet, or 4096 units, which is the length of a cell (4096 x 4096).

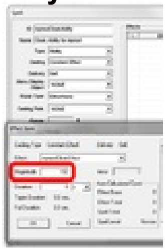
- Cloak Effect
- Cloak Ability



## Create the Applying Spell and Effect

The Applying spell will be cast on an Actor when he/she comes into contact with the Cloak. This spell is an intermediary between the Cloak and Actor, and in its simplest form only needs to run AddSpell().

First we will set up the effect which we will put onto the Applying Spell. In the **Object Window**, navigate to **Magic Effects**, under the Magic category. Create a new Magic Effect by right-clicking on the list and choosing 'New'. For this effect, change the **Effect Archetype** to `Script`, the **Casting Type** to `Concentration` and **Delivery** to `Aimed`.



Important Flags

> **No Death Dispel** - Needed if we want this effect to affect dead actors as well. In general, it is a good idea to check this and handle the cleanup of a magic effect inside an OnDying() Event. [1]
>
> **No Hit Event** - We *must* check this flag as one of the steps to avoid the **'Brawl Bug'**.

We'll give this the ID **`ApplyingEffect`** and a descriptive Name, and then click OK to accept the changes. Open the Effect again to add a new a script, which we will title **`ApplyingScript`**:

```
Scriptname ApplyingScript extends ActiveMagicEffect

Spell Property MonitorAbility Auto

Event OnEffectStart(Actor akTarget, Actor akCaster)
 akTarget.AddSpell(MonitorAbility)
EndEvent
```

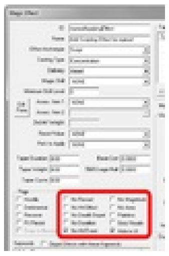
Save the script, and click OK to accept all changes. We will be coming back to this effect to fill its Conditions and script properties later.

In the **Object Window**, navigate to **Spells** under the Magic category. Create a new Spell by right-clicking on the list and choosing 'New'. Keep the Type as `Spell`, but change the Casting Type to `Concentration` and set Delivery to `Aimed`. We'll give this the ID **`ApplyingSpell`** and a descriptive Name. In the Effects list, we will add **`ApplyingEffect`**. Leave magnitude at 0 and leave the Conditions list blank. Click OK to accept all changes.

**IMPORTANT:** To ensure the **`ApplyingSpell`** is correctly applied to targets, you may need to change the duration of the **`ApplyingSpell`** to 1 (instead of 0, as it appears in the images).

## Attaching the Spell to our Cloak Effect

With **`ApplyingSpell`** saved, we now must attach it to our Cloak Effect. Open **`CloakEffect`** and in the Assoc. Item 1 dropdown, find and select **`ApplyingSpell`** from the list. Click OK to accept changes on **`CloakEffect`**



- Applying Effect
- Applying Spell
- Attaching the Spell

## Create the Ability and Effect for the Actor

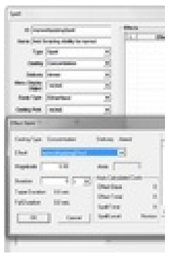
This is the spell that will be applied to the Actors. Again, for the purposes of this tutorial we have chosen to attach a script to Actors which monitors the damage done to them by the player.

Create a new magic effect like before, and set its Effect Archetype to `Script`, its Casting Type to `Constant Effect` and Delivery to `Self`.

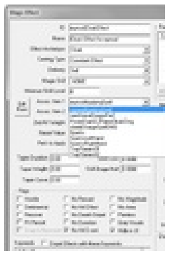
Important Flags

> **No Death Dispel** - Needed if we want this effect to affect dead actors as well. In general, it is a good idea to

check this and handle the cleanup of a magic effect inside an OnDying() Event. [1]
**No Hit Event** - We *must* check this flag as one of the steps to avoid the **'Brawl Bug'**.



We'll be giving it the ID **`MonitorEffect`** and a descriptive Name, then click OK to accept the changes. Open the Effect again to add a new script, which we will title **`MonitorScript`**:

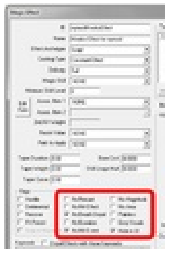


```
Scriptname MonitorScript extends ActiveMagicEffect

Actor MySelf
Float Health

Event OnEffectStart(Actor akTarget, Actor akCaster)
 MySelf = akTarget
 Health = MySelf.GetActorValue("Health")
 RegisterForSingleUpdate(0.25)
EndEvent

Event OnUpdate()
 Health = MySelf.GetActorValue("Health")
 RegisterForSingleUpdate(0.25)
EndEvent

Event OnEffectFinish(Actor akTarget, Actor akCaster)
 UnregisterForUpdate()
EndEvent

Event OnHit(ObjectReference akAggressor, Form akSource, Projectile akProjectile, bool
abPowerAttack, \
  bool abSneakAttack, bool abBashAttack, bool abHitBlocked)
 Float Damage = Health - MySelf.GetActorValue("Health")
 Health = MySelf.GetActorValue("Health")
 if (akAggressor == Game.GetPlayer()) && (Damage > 0)
  Debug.Notification("You did " + Damage as Int + " points of damage.")
 endif
EndEvent
```

Save the script, and click OK to accept all changes. We will come back to this effect to fill its Conditions later.

Create a new spell like before. We'll give this the ID **`MonitorAbility`** and a descriptive Name. Change the Effect Archetype to `Ability`, set Casting Type to `Constant Effect` and Delivery to `Self`. In the Effects list, we will add **`MonitorEffect`**. Leave Magnitude at 0 and leave the Conditions list blank. Click OK to accept the changes.

- Monitor Effect
- Monitor Ability



## Putting It All Together

### Filling Properties

First we need to fill the **MonitorAbility** property on the **ApplyingEffect** and its script. Open **ApplyingEffect**, then open its **Properties window** and auto-fill the `MonitorAbility` property, or select MonitorAbility manually from the drop-down menu.

## Filling Conditions

Now we want to conditionalize some of the effects so that they only get applied when they should. With **ApplyingEffect** still open, right-click in the Target Conditions list and select 'New'. Our first condition will make it such that it does not attempt to apply **MonitorEffect** to an Actor if he/she already has the effect:

```
HasMagicEffect MagicEffect: 'MonitorEffect' == 0
```

This needs to be run on **Subject**. Optionally, instead of HasMagicEffect we may use the more general HasMagicEffectKeyword with our own custom Keyword, especially if we want to apply more than one type of Magic Effect to an Actor based on certain conditions. This is covered below in **Multiple Conditional Magic Effects**.

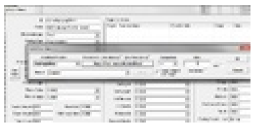
We may also not want the spell to apply our script to dead Actors, so we should add another condition to the **Subject**:

```
GetDead NONE == 0
```

You can also reorder your condition functions by clicking **<<** or **>>** with the condition selected. It is more important that the Actor does not have **MonitorEffect** so we can place this above GetDead. The game *should* stop condition testing the when the first condition tests false, but it is unknown if it does so, so consider this a minor optimization.

We shall also apply some conditions to the CloakEffect, but we will cover this in the section covering the **'Brawl Bug'**

Note that by adding a **Condition Function** to the Magic Effect itself the condition test is only run once -- when the Magic Effect is first applied. To test continuously for a condition, use the Target Conditions list for the Magic Effect on the Spell instead.

- 

  Condition 1

-

  Condition 2

## Adding the Cloak to the `PlayerAlias`

The only thing left to do is add our **CloakAbility** to our **PlayerAlias**. There are two options: We can either add the spell to the Alias Spells list; or if we are interested in **Reducing Cloaking Frequency** and circumventing the

'**Brawl Bug**', we can follow the steps outlined below.

Now start the game with the mod loaded and attack someone. The damage dealt by the player should be seen as messages on the upper left corner of the screen.

## Optimizations & Bug Fixes

In this section we will deal with some of the downsides of this method, and ways of working around them. The most significant of which is the '**Brawl Bug**'.

### Reducing Cloaking Frequency

If your script is like the example shown here, whereby most of the logic is occuring within an **OnUpdate()** event on each Actor, we can significantly reduce the time spent cloaking surrounding NPCs. In our effect, we are using a cloaking radius of `192.0`, which equates to a cell's width on all sides of the player. It takes the player character about 8-10 seconds to sprint from one cell border to another, so we will attempt to run the cloak at least every half cell width, or every 4-5 seconds. So, by the time we run a half cell away, our cloak effect will have already applied the spell to Actors a full cell ahead of the player character.

We can accomplish this by adding the **`CloakAbility`** to the player, waiting a short amount of time, and then removing it. To do this, we will need to add a script to our **`PlayerAlias`**. Open your quest, and click on the **Quest Alias** tab. Double-click on our **`PlayerAlias`** to open up the Reference Alias window. Add a new script to our alias. We will call it **`PlayerScript`**, and fill it with the following code:

```
Scriptname PlayerScript extends ReferenceAlias

Spell Property CloakAbility Auto
Actor Property PlayerRef Auto

Event OnInit()
 RegisterForSingleUpdate(1)
EndEvent

Event OnUpdate()
 PlayerRef.AddSpell(CloakAbility, false)
 ; How long you would like to keep the cloak active
 Utility.Wait(1)
 PlayerRef.RemoveSpell(CloakAbility)
 ; How long until the cloak activates again
 RegisterForSingleUpdate(4)
EndEvent
```

Now we will save the script and auto-fill the properties. This method of adding and removing the cloak is also used in circumventing the '**Brawl Bug**'.

### Death Dispel

Not using the **No Death Dispel** flag on effects with scripts can lead to errors.[1] To avoid these, you should handle the cleanup of your magic effect in the magic effect itself. Example:

```
Scriptname MonitorScript extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
 ; Any Initialization/Setup code here
 ; Go to our Alive state
 GoToState("alive")
EndEvent

State alive

 Event OnBeginState()
  RegisterForSingleUpdate(0.25)
 EndEvent

 Event OnUpdate()
  ; Regular code here
  RegisterForSingleUpdate(0.25)
 EndEvent

 Event OnDying(Actor akKiller)
  ; Any additional cleanup code here
  UnregisterForUpdate()
  GoToState("dead")
 EndEvent

EndState

State dead
; Nothing needed here
EndState
```

## The 'Brawl Bug'

The 'Brawl Bug' affects several quests which rely on determining if the player has cast magic on the Actor. This means that our **`CloakEffect`** can inadvertently progress various quests to stages which we may not desire. The most prominent example of these undesired effects is during Brawls, where using anything but fists will cause the fight to become "real".

We have already taken one step toward circumventing this issue, and that is by checking `No Hit Event` on our magic effects. Without this, none of our other changes will matter, and it will also cause the player to receive bounty in addition to causing the fight to turn sour.

First we need to add some Condition Functions to our **`CloakAbility`** and **`CloakEffect`**. Open our **`CloakAbility`** and double click on its magic effect. Adding conditions to the magic effect here will cause the effect to dispel the moment these conditions become false. We should add the following conditions:

```
GetQuestRunning Quest: 'DGIntimidateQuest' == 0
GetStage Quest: 'C00' != 20
```

To elaborate, `DGIntimidateQuest` is the quest that begins running when a Brawl starts, and stops running when it ends. Quest `C00` is the main Companions quest line, and specifically we are disallowing it from running during the stage where you must fight Vilkas. Note, this is technically not one of the game-breaking examples of the bug, since Vilkas only needs to be punched three times to end the stage and progress the Companions quest. What this condition does is prevent him from commenting on the player's "magic use".

Unfortunately, there is a minute chance that at the start of a Brawl our **`CloakEffect`** will still be active, and not dispel quickly enough. To deal with this, we can also pause our cloak during some of the sub-quests leading up to the Brawl. A non-exhaustive list:

```
GetQuestRunning Quest: 'Favor017' == 0
GetStage Quest: 'FreeformRiften19' != 30
```

And if all else fails...

## Include an Off Switch

In case a user is having issues with the cloaking effects from our mod, we will provide an off switch to allow the user to temporarily cirumvent the issue without uninstalling our mod. We will simply create a new **GlobalVariable** and wrap our Add/RemoveSpell logic on **`PlayerScript`** with it. To create a new global, in the **Object Window**, navigate to **Global** under the Miscellaneous category. In the object list, right-click and select 'New' to create a new global variable. We will call this **`mymod_CloakEffectOn`** (replace "mymod" with your own prefix) and set the value to `1`. We will leave the Variable Type as `Short`.

Now we will wrap our code in **`PlayerScript`** with a condition to run only when this global's value is set to `1`:

```
Scriptname PlayerScript extends ReferenceAlias

Spell Property CloakAbility Auto
Actor Property PlayerRef Auto
GlobalVariable Property mymod_CloakEffectOn Auto

Event OnInit()
 RegisterForSingleUpdate(1)
EndEvent

Event OnUpdate()
 If mymod_CloakEffectOn.GetValue()
  PlayerRef.AddSpell(CloakAbility, false)
  ; How long you would like to keep the cloak active
  Utility.Wait(1)
  PlayerRef.RemoveSpell(CloakAbility)
 EndIf
 ; How long until the cloak activates again
 RegisterForSingleUpdate(4)
EndEvent
```

Don't forget to auto-fill the new **GlobalVariable** property on our **`PlayerScript`**. Users now have the opportunity of pausing the cloak on their own with the console command `set mymod_CloakEffectOn to 0`.

### Multiple Conditional Magic Effects

Placeholder. (This section is a WIP)

## References

1. See Common Errors.
2. Brawl Bug Modder's Resource

---

## Attaching Scripts to Objects

Unlike with actors, objects cannot run magic effect scripts. To give an object a script at runtime, we need to assign the object a reference alias with a reference alias script. For this tutorial, we will create a mod that gives the player firewood whenever he/she hits a tree with an axe.

### Create a FormList

First, we're going to create a FormList of trees that we want to give out firewood. We will be using this later to fill the reference aliases we create.

Navigate to **FormList** under **Miscellaneous** in the **Object Window**. Right-click on the list and select 'New' to create a new formlist. Now navigate to **Landscape** under **Tree** which is under **World Objects** in the **Object Window**. Highlight everything in the list, then click and drag them into the new formlist.

Click OK to create the formlist. Hereafter we will refer to this formlist as **TreesList**.

### Create the First Quest

This quest is going to be holding the reference aliases that we will be assigning to objects at runtime. Navigate to **Quest** under **Character** in the **Object Window**. Right-click on the list and select 'New' to create a new quest.

Go to the **Quest Alias** tab, then right-click on the empty list and choose to create a **New Reference Alias**. Check the flag for **Optional**. For **Fill Type**, choose **Find Matching Reference** and check the flags for **In Loaded Area** and **Closest**. Under **Match Conditions**, right-click the list and select 'New' to create a new condition:

```
IsInList 'TreesList' == 1.00
```

Click OK to close the alias window. Then double-click on the alias in order to open it again. We need to do this step because when we first create an alias, we can't attach a script to it. Attach this script to the reference alias:

```
Scriptname TreeAliasScript extends ReferenceAlias

Keyword Property WeapTypeBattleAxe Auto
Keyword Property WeapTypeWarAxe Auto
MiscObject Property Firewood01 Auto

Event OnHit(ObjectReference akAggressor, Form akSource, Projectile akProjectile, bool
abPowerAttack, \
  bool abSneakAttack, bool abBashAttack, bool abHitBlocked)
 if (akAggressor == Game.GetPlayer())
  if (akSource.HasKeyword(WeapTypeBattleAxe) || akSource.HasKeyword(WeapTypeWarAxe))
   akAggressor.AddItem(FireWood01)
  endif
 endif
EndEvent
```

Set the properties for the script. Since the properties are named exactly the same as the editor IDs of what we want them to represent, just click the **Auto-Fill All** button to set all the values automatically.

Click OK to close the alias window. Now right-click on the reference alias we just created and select 'Duplicate' to create a duplicate of the alias. You can do this as many times as you like, although five is probably enough for most circumstances.

Click OK to close the quest window. Hereafter we will refer to this quest as **DASQuest**.

## Create the Second Quest

This quest is going to start and stop the first quest. We do this so that the game will constantly refill the aliases in the first quest with the closest trees.

Navigate to **Quest** under **Character** in the **Object Window**. Right-click on the list and select 'New' to create a new quest. Click OK to close the quest window, then double-click the quest to open the window again. This step is necessary because if we try to attach a script to a quest right when we create it, the CK crashes (at least in the author's experience).

Attach this script to the quest:

```
Scriptname UpdateQuestScript extends Quest

Quest Property DASQuest Auto

Event OnInit()
 RegisterForSingleUpdate(5)
EndEvent

Event OnUpdate()
 DASQuest.Stop()

        ;We must wait until the quest has stopped before restarting the quest. Using
Stop() and immediately Start() may fail.
        ;Continue after 5 seconds to prevent infinite loop.
        int i = 0
        while !DASQuest.IsStopped() && i < 50
            Utility.Wait(0.1)
            i += 1
        endWhile

 DASQuest.Start()
 RegisterForSingleUpdate(5)
```

```
EndEvent
```

Remember to set the value of the 'DASQuest' property to the **DASQuest** we created earlier. Check to make sure that **Start Game Enabled** is checked in the **Quest Data** tab, then click OK to close the quest window.

### Conclusion

The two quests are all that's necessary for this mod. Save the mod and then load the game with it to test it out. A piece of firewood should be received every time the player hits a tree with an axe.

The method for attaching scripts to objects using reference aliases is much easier than what we did with magic effects and actors. But there are disadvantages to using it:

1. The number of actors/objects that we can attach the script to is limited to the number of reference aliases we create.
2. The script does not stay on the actor/object because we need to constantly refill the aliases.

So while this method worked for what we did here, it might not be good for doing something else. For example, what if instead we wanted the trees to only give out firewood after being hit three times instead of once?

That would have been much more complicated because then we would have to store the number of times we hit the tree somewhere. Obviously, we cannot store this variable in the reference alias scripts because the object filling the alias is not always going to be the same tree. We might swing our axe twice but then the alias gets refilled before the third strike, which means it could take us 5 strikes to get a piece of firewood.

## Known Issues

- When using the second method with constantly refilling aliases, be careful with your alias scripts. If the alias script happens to be running a while loop when its quest is stopped, then it leads to suspended stacks, in turn leading to suspended scripts.

## External Links

Porting Kuertee's Sittable Rocks Mod from Oblivion

Dynamically Attaching Scripts to Actors Near the Player

Suspended Stacks Suspends Your Scripts, Sometimes Indefinitely