Dissecting the Scripts for Weapon Racks

creationkit.com/index.php

Contents

Did you ever wonder just how weapon plaques work? How does clicking on one lead to your sword hanging on the wall? Well now you can find out.

A weapon plaque is composed of two things, a weapon rack trigger, and a weapon rack activator. The weapon rack trigger is the part that you can see, such as the rack itself or the plaque. The weapon rack activator is the part that you don't see but which prompts you for activation.

Weapon Rack Trigger

The weapon rack trigger uses the script 'WeaponRackTriggerSCRIPT' so let's open that up and take a look.

Properties

The first couple lines are the properties declared for this script. None of these are really important in understanding how the script works, so we'll move on.

OnReset

```
EVENT OnReset()
 AlreadyInit = FALSE
 ;Trace("DARYL - " + self + " running OnReset() and AlreadyInit = " + AlreadyInit)
endEVENT
```

The first event that's in the script is the OnReset event. The code triggers when the object is reset. Here, it is used to reset the variable 'AlreadyInit'. You'll see what the variable is actually used for in the next event. (Even though they are properties, you may find that I call them variables because properties are basically just a subtype of variables.)

OnLoad

```
EVENT OnLoad()
 ;Trace("DARYL - " + self + " running OnLoad() and AlreadyInit = " + AlreadyInit)
 if (AlreadyInit == FALSE) && (self.IsEnabled())
  ActivatorRef = GetLinkedRef(WRackActivator)
  if (ActivatorRef)
   ActivatorRef.Enable()
  endif
  AlreadyInit = TRUE
  ;Trace("DARYL - " + self + " The Activator Ref is " + ActivatorRef)
 else
  ;Do nothing
 endif
```

```
 ;Trace("DARYL - " + self + " finishing OnLoad() and AlreadyInit = " + AlreadyInit)
endEVENT
```

The next event is the OnLoad event. The OnLoad block is triggered when the object's model is loaded (ie it's rendered). If 'AlreadyInit' is false, and the object itself is enabled, then it tries to get a reference to its linked ref with keyword associated with the property 'WRackActivator'. If there is a linked ref, it enables it. Whether there is a linked reference or not, the 'AlreadyInit' variable gets set to true, so it doesn't try to do this whole sequence again until the object gets reset.

OnCellLoad

```
;EVENT OnCellLoad()
; ;Trace("DARYL - " + self + " running OnCellLoad() and AlreadyInit = " +
AlreadyInit)
; if (AlreadyInit == FALSE) && (self.IsEnabled())
;  ActivatorRef = GetLinkedRef(WRackActivator)
;  AlreadyInit = TRUE
;  ;Trace("DARYL - " + self + " The Activator Ref is " + ActivatorRef)
; else
;  ;Do nothing
; endif
; ;Trace("DARYL - " + self + " finishing OnCellLoad() and AlreadyInit = " +
AlreadyInit)
;endEVENT
```

The OnCellLoad event is actually totally commented out, so it won't run in the game. But it looks like they tried to get it to do the same thing as the OnLoad block. They didn't get to enabling the linked ref though.

OnTriggerEnter and OnTriggerLeave

Next, you see the OnTriggerEnter and OnTriggerLeave blocks inside the 'WaitingForReference' state. I'm not going to try and explain states here, but it's redundant and has no utility if you only have a single state in a script.

OnTriggerEnter

```
EVENT onTriggerEnter(objectReference triggerRef)
 ;Trace("DARYL - " + self + " Just OnTriggerEntered by " + triggerref)

 if (IgnoreArmor == TRUE) && (triggerRef.GetBaseObject() as Armor)
  ;Trace("DARYL - " + self + " I'm ignoring armor, and this is armor, so I'm doing
nothing.")
  ;Do Nothing
 else
  ;Trace("DARYL - " + self + " I'm updating as normal")
  ;numInTrig = (numInTrig + 1)
  HasBeenTriggered = TRUE
  if ActivatorRef
   ActivatorRef.Disable()
  endif
 endif
endEVENT
```

The OnTriggerEnter block is triggered when something enters the trigger area of the object. What it's doing here is to disable the linked ref. That's what makes it so that you can't put another weapon on a weapon plaque when you already have one there. The variable 'HasBeenTriggered' is just to make sure it doesn't keep triggering until that variable is reset.

OnTriggerLeave

```
EVENT OnTriggerLeave(objectReference triggerRef)
 ;Trace("DARYL - " + self + " A References has EXITED, refs in this trigger now = "
+ numInTrig)

 if (IgnoreArmor == TRUE) && (triggerRef.GetBaseObject() as Armor)
  ;Trace("DARYL - " + self + " I'm ignoring armor, and this is armor, so I'm doing
nothing.")
  ;Do Nothing
 else
  ;Trace("DARYL - " + self + " I'm updating as normal")
  ;numInTrig = (numInTrig - 1)
 endif

 if (GetTriggerObjectCount() == 0)
  HasBeenTriggered = FALSE
  ActivatorRef.Enable()
 endif

endEVENT
```

The OnTriggerLeave block is triggered when something leaves the trigger area of the object. The code here is to enable the linked ref again. This is what allows you to put a new weapon on the plaque when you take the old one off. And 'HasBeenTriggered' is here again to make sure that it only triggers once until the variable is reset.

Conclusion

So after going through the script, we can say that its only purpose is to control whether or not its linked ref is enabled or disabled, depending on whether or not there's something in its own trigger area.

Weapon Rack Activator

The weapon rack activator uses the script 'WeaponRackActivateSCRIPT' so lets open that up and take a look.

Properties

The first 100 or so lines are filled with properties. The important one that we need to look at is the 'RackType' property.

```
int Property RackType = 1 Auto
{The type of rack this script is on:  Default = 1
1 = Standard Weapon Rack (Includes Regular, Mount, and CoA Weapon Racks)
2 = COA Shield Rack
```

```
3 = COA Weapon Rack (Both left and right)
4 = Table-top Dagger Rack
}
```

As the comment itself will tell you, this property is used to distinguish which object the script is on. This is necessary because this script is used for the weapon racks, the weapon plaques, and the display cases.

OnCellLoad

This is the code for the event OnCellLoad, without most of the comments:

```
EVENT OnCellLoad()
 Trace("DARYL - " + self + " running OnCellLoad() and AlreadyInit = " + AlreadyInit)
 TriggerMarker = GetLinkedRef(WRackTrigger)
 Trace("DARYL - " + self + " The TriggerMarker is " + TriggerMarker)
 if (TriggerMarker) && (AlreadyInit == FALSE) && (TriggerMarker.IsEnabled())
  StartingWeapon = GetLinkedRef()
  Trace("DARYL - " + self + " The Starting Weapon is " + StartingWeapon)
  TriggerMarker = GetLinkedRef(WRackTrigger)
  Trace("DARYL - " + self + " The TriggerMarker is " + TriggerMarker)

  if (StartingWeapon)
   if StartingWeapon.Is3DLoaded()
    if StartingWeapon.GetParentCell() == self.GetParentCell()
     Trace("DARYL - " + self + " Has a starting weapon")
     HandleStartingWeapon()
    endif
   endif
  else
   Trace("DARYL - " + self + " Doesn't have a starting weapon")
   ;Do nothing
  endif

  AlreadyInit = TRUE
 else
  ;Do nothing
 endif
 Trace("DARYL - " + self + " finishing OnCellLoad() and AlreadyInit = " +
AlreadyInit)
endEVENT
```

The comments that have been omitted are not relevant, not used in the rest of the script, and probably from an early and abandoned attempt of making the script do what it does.

Here, we see that when the cell is loaded, the first thing the script does is try to assign something to 'StartingWeapon'. It looks for any linked ref without a keyword associated with it. Then the script attempts to locate the linked ref with the keyword associated with property 'WRackTrigger' and assign it to 'TriggerMarker'.

If it manages to locate this 'TriggerMarker', and this portion of the script has not been run before (AlreadyInit == False), and the 'TriggerMarker' happens to be enabled, it checks to see if anything had been assigned to 'StartingWeapon'.

If there is a starting weapon, then it checks to make sure that the weapon is loaded, and that it happens to be in the same cell as the activator. If all this is true, it calls the function 'HandleStartingWeapon'. The code for that is further down the script, and we will get to that later.

After all of this, it will set 'AlreadyInit' to true, so that this code is not run again.

OnActivate

The OnActivate block is very long, so we're going to look at it in sections. You might notice that the whole block is encased in an "Empty Rack" state, but again this is pointless when you only have one state in the script.

Standard Rack

```
If (RackType == 1)
; This is the Standard Weapon Rack

 if (TriggerRef == Game.GetPlayer() as Actor)
 ; Only the player can activate this

 MessageAlreadyShown = WRackGlobal.GetValue()

  if (MessageAlreadyShown == FALSE)
   ; If the help message hasn't been shown before then show it.
   WeaponRackActivateMESSAGE.Show()
   WRackGlobal.SetValue(1)
  else
   Trace("DARYL - " + self + " Player activated the weapon rack")
   PlayersEquippedWeaponType = Game.GetPlayer().GetEquippedItemType(1)
   ; Get the EquippedItemType that is in the players right hand

   if (PlayersEquippedWeaponType == 0) || (PlayersEquippedWeaponType == 9) ||
(PlayersEquippedWeaponType == 10) || (PlayersEquippedWeaponType == 11)
   ; If the player is unarmed, or has a spell/shield/torch equipped, tell him he
needs a weapon equipped.
    WeaponRackNoWeaponMESSAGE.Show()

   ;elseif (PlayersEquippedWeaponType == 2)
   ; If the player has a dagger equipped tell him it doesn't fit.
    ;WeaponRackNoDaggerMESSAGE.Show()

   else

    HandleWeaponPlacement()
    ; Grabs the weapon from the player and places it in the correct place.

   endif

  endif

 endif
```

The first if statement here is to check that the object the script is attached to is the standard weapon rack. The standard weapon racks include the standing racks, the plaque that only holds one weapon, and the large display case.

The second if statement is to make sure that it's being activated by the player. The third is to check whether or not this is the player's first time activating a weapon rack, in which case it would show a messagebox explaining how to use one.

If it isn't the player's first time using a weapon rack, then it will assign the item type equipped in the player's right hand to 'PlayersEquippedWeaponType'. If you don't know the values for the weapon types, you should take a look at GetEquippedItemType.

If the player is unarmed, or else equipped with a spell, it would show the message that the player needs to have a weapon equipped in order to use the weapon rack. There's a part here that's commented out. Looks like it was originally not possible to put daggers on the standard weapon racks but then they changed it. If the player does have a weapon equipped, it calls the function 'HandleWeaponPlacement' which is defined further down the script.

Shield Rack

```
elseIf (RackType == 2)
 ;This is the CoA Shield Rack

 if (TriggerRef == Game.GetPlayer() as Actor)
 ; Only the player can activate this

 MessageAlreadyShown = WRackGlobal.GetValue()

  if (MessageAlreadyShown == FALSE)
   ; If the help message hasn't been shown before then show it.
   WeaponRackActivateMESSAGE.Show()
   WRackGlobal.SetValue(1)
  else
   Trace("DARYL - " + self + " Player activated the shield rack")
   Trace("DARYL - " + self + " Player has shield " +
Game.GetPlayer().GetEquippedShield() + " base object equipped")
   ; PlayersEquippedWeaponType = Game.GetPlayer().GetEquippedItemType(0)
   ; Get the EquippedItemType that is in the players left hand

   if (Game.GetPlayer().GetEquippedShield())
   ; Grabs the weapon from the player and places it in the correct place.
   HandleWeaponPlacement()


   else
   ; If the player doesn't have a shield equipped tell him he needs one.
    WeaponRackNoShieldMESSAGE.Show()


   endif
```

```
  endif

 endif
```

Continuing from what we had above, this is for the case when the object that the script is attached to is the shield rack.

Again, the next two things that the script checks for is that the player was the one who activated the object, and whether or not it's the first time that the player has activated a weapon rack. The same thing is done for all the rack types, so it would have been a better idea to check for those before checking what kind of rack is being activated.

If the player has seen the message before, then the script will check if the player has a shield equipped. If so, the script calls the function 'HandleWeaponPlacement' and if not, it will show the message stating that you need to have a shield equipped to use the shield rack.

COA Weapon Rack

```
elseIf (RackType == 3)
; This is the COA Weapon Rack

 if (TriggerRef == Game.GetPlayer() as Actor)
 ; Only the player can activate this

 MessageAlreadyShown = WRackGlobal.GetValue()

  if (MessageAlreadyShown == FALSE)
   ; If the help message hasn't been shown before then show it.
   WeaponRackActivateMESSAGE.Show()
   WRackGlobal.SetValue(1)
  else
   Trace("DARYL - " + self + " Player activated the weapon rack")
   PlayersEquippedWeaponType = Game.GetPlayer().GetEquippedItemType(1)
   ; Get the EquippedItemType that is in the players right hand

   if (PlayersEquippedWeaponType == 0) || (PlayersEquippedWeaponType == 9) ||
(PlayersEquippedWeaponType == 10) || (PlayersEquippedWeaponType == 11)
   ; If the player is unarmed, or has a spell/shield/torch equipped, tell him he
needs a weapon equipped.
    WeaponRackNoWeaponMESSAGE.Show()

   elseif (PlayersEquippedWeaponType == 2)
   ; If the player has a dagger equipped tell him it doesn't fit.
    WeaponRackNoDaggerMESSAGE.Show()

   elseif (PlayersEquippedWeaponType == 7)
   ; If the player has a bow equipped tell him it doesn't fit.
    WeaponRackNoBowMESSAGE.Show()

   else
```

```
    HandleWeaponPlacement()
    ; Grabs the weapon from the player and places it in the correct place.

   endif

  endif

 endif
```

This section deals with COA weapon racks. COA stands for Coat Of Arms, and these are the plaques that allow you to have a weapon on the left and right sides.

The first thing it checks is that this is a COA weapon rack by looking at its RackType property.

- The RackType property is supposed to be 3 for all of the COA weapon racks, but whoever placed the script on WeaponRackCOALeftACTIVATORPlayerHouse and WeaponRackCOARightACTIVATORPlayerHouse forgot to edit the property and change it from the default of 1, which is why you might end up with floating bows and daggers when using them.

Then it checks if the player is the one who activated the weapon rack, and whether or not it's the player's first time doing so and therefore it should show the messagebox. Otherwise, it then assigns the item type of whatever is equipped in the player's right hand to the 'PlayersEquippedWeaponType' variable.

If the player is unarmed, holding a torch, shield, or spell, it will show the message that you need to have a weapon equipped. If the player is holding a bow or dagger, it will show the message that the weapon doesn't fit on the rack. Otherwise, it will call the function 'HandleWeaponPlacement'.

Dagger Display

```
elseIf (RackType == 4)
; This is the Standard Weapon Rack

 if (TriggerRef == Game.GetPlayer() as Actor)
 ; Only the player can activate this

 MessageAlreadyShown = WRackGlobal.GetValue()

  if (MessageAlreadyShown == FALSE)
   ; If the help message hasn't been shown before then show it.
   WeaponRackActivateMESSAGE.Show()
   WRackGlobal.SetValue(1)
  else
   Trace("DARYL - " + self + " Player activated the weapon rack")
   PlayersEquippedWeaponType = Game.GetPlayer().GetEquippedItemType(1)
   ; Get the EquippedItemType that is in the players right hand

   if (PlayersEquippedWeaponType != 2)
   ; If the player is trying to place anything but a dagger tell him he can't.
    WeaponRackOnlyDaggerMESSAGE.Show()
```

```
   else

    HandleWeaponPlacement()
    ; Grabs the weapon from the player and places it in the correct place.

   endif

  endif

 endif
```

The last part of the OnActivate block deals with the case that RackType is 4, or that the rack is a dagger display case. It goes through the standard checks to see if it's the player and if it's your first time using a weapon rack.

Then it checks for what's equipped in the player's right hand. If the player is equipped with anything but a dagger, it will show the message that you can only place a dagger there. If the player is equipped with the dagger, then the script calls the function 'HandleWeaponPlacement'.

You might notice that in all these cases, the weapon is assumed to be in the right hand. That means that the script does not try and detect weapons in the player's left hand, which is why you can't place any weapons you have equipped in your left hand onto any weapon rack.

HandleWeaponPlacement

Let's look at the HandleWeaponPlacement function in two parts. The first part:

```
Function HandleWeaponPlacement(bool ForStartingWeapon = FALSE)
 ; Grabs the weapon from the player and places it in the correct place.

 if (ForStartingWeapon)
  Trace("DARYL - " + self + " Handling Starting Weapon")
  ;Don't do the player stuff
 else
  Trace("DARYL - " + self + " Handling Players Weapon")

  If (RackType == 2)
   Trace("DARYL - " + self + " Player currently has weapon type " +
PlayersEquippedWeaponType + " in his left hand.")
   ;PlayersEquippedShield = Game.GetPlayer().GetEquippedWeapon(TRUE)
   ; Find out what shield the player currently has equipped

   Trace("DARYL - " + self + " Player has shield " +
Game.GetPlayer().GetEquippedShield() + " base object equipped")
   PlayersDroppedWeapon =
Game.GetPlayer().DropObject(Game.GetPlayer().GetEquippedShield(), 1)
   ; Force the weapon to be dropped, and get it's reference

   Trace("DARYL - " + self + " Dropped " + PlayersEquippedShield + " shield from
players inventory as " + PlayersDroppedWeapon)
  else
   Trace("DARYL - " + self + " Player currently has weapon type " +
```

```
PlayersEquippedWeaponType + " in his right hand.")
   PlayersEquippedWeapon = Game.GetPlayer().GetEquippedWeapon()
   ; Find out what weapon the player currently has equipped

   Trace("DARYL - " + self + " Player has " + PlayersEquippedWeapon + " base object
equipped")
   PlayersDroppedWeapon = Game.GetPlayer().DropObject(PlayersEquippedWeapon, 1)
   ; Force the weapon to be dropped, and get it's reference

   Trace("DARYL - " + self + " Dropped " + PlayersEquippedWeapon + " from players
inventory as " + PlayersDroppedWeapon)
  endif

  if (PlayersDroppedweapon)
   int Count = 0
   While(!PlayersDroppedweapon.Is3DLoaded()) && (Count < 10)
    ; Have to wait to make sure the item is dropped before setting it's motion type,
or else I get a "Object has no 3D" error.
    Utility.Wait(0.1)
    Count += 1
   EndWhile
  endif

 endif
```

This part of the function is to make the player drop the his/her weapon. First it checks if there's a starting weapon already assigned to the weapon rack. If there is, then that means this function wasn't called becaues the player activated the rack.

If there isn't a starting weapon, then it checks to see if the weapon rack is type 2 (shield rack). If it's a shield rack, it will make the player drop his/her shield. Otherwise, it will make the player drop his/her weapon. It will also assign the reference to the dropped object to the variable 'PlayersDroppedWeapon'.

> The DropObject function will first look through the player's inventory for non-equipped items. Only if it finds none will it try to drop any equipped items. The RemoveItem function will first look through the equipped items. Since this script uses DropObject, it may not always grab the player's equipped weapon/shield. However, this is understandable because to use RemoveItem will require adding an extra container to the whole process.

The last part of the code above is to make pause the script until the weapon/shield actually shows up outside of your inventory.

The second part of the HandleWeaponPlacement function:

```
 PlayersDroppedWeapon.SetMotionType(Motion_Keyframed, false)
 ; Tell the weapon to ignore all forms of physic interaction
 Trace("DARYL - " + self + " Disabling physics on " + PlayersDroppedWeapon)

 ; Handle the placement of the weapon
 if PlayersDroppedWeapon.HasKeyword(WeaponTypeSword)
  ; 1H Sword
```

```
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
SwordMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "SwordPivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeWarAxe)
  ; 1H Axe
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
WaraxeMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "WarAxePivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeMace)
  ; Mace
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
MaceMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "MacePivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeGreatSword)
  ; 2H Sword
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
GreatSwordMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "GreatSwordPivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeBattleAxe)
  ; 2H Axe
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
BattleaxeMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "BattleAxePivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeWarhammer)
  ; Warhammer
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
WarhammerMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "WarhammerPivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeBow)
  ; Bow
  Trace("DARYL - " + self + " Moving " + PlayersEquippedWeaponType + " to the
BowMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "BowPivot01")


 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeStaff)
  ; Staff
  Trace("DARYL - " + self + " Moving players dropped weapon" + PlayersDroppedWeapon +
```

```
" of type " + PlayersEquippedWeaponType + " to the StaffMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "StaffPivot01")

 elseif PlayersDroppedWeapon.HasKeyword(WeaponTypeDagger)
  ; Staff
  Trace("DARYL - " + self + " Moving players dropped weapon" + PlayersDroppedWeapon +
" of type " + PlayersEquippedWeaponType + " to the DaggerMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "DaggerPivot01")

 elseif PlayersDroppedWeapon.HasKeyword(ArmorShield)
  ; Staff
  Trace("DARYL - " + self + " Moving players dropped weapon" + PlayersDroppedWeapon +
" of type " + PlayersEquippedWeaponType + " to the ShieldMarker")
  PlayersDroppedWeapon.MoveToNode(TriggerMarker, "ShieldPivot01")


 endif
EndFunction
```

This part of the function is the part that actually deals with hanging your shield/weapon onto the rack/plaque/display case. First it set the motion type to key-framed, which means that it is unaffected by Havok physics. Then it checks what kind of weapon/shield it is through the use of keywords. All the vanilla bows have the keyword WeapTypeBow, all shields have the keyword ArmorShield, etc. Once the script has determined the type of weapon it is, then it will use the MoveToNode function to hang the weapon onto the linked ref (the actual rack/plaque/display case).

This is the reason why it's important that the RackType property is set correctly, because not all of the displays have the same nodes. If you try to hang a weapon onto a node that doesn't exist, you end up with a floating weapon.

You might be thinking, "But wait! What about the starting weapon? This part of the code doesn't deal with that!" That's covered by the next function, 'HandleStartingWeapon'.

HandleStartingWeapon

```
Function HandleStartingWeapon()

 If (RackType == 1)
  ; If this is a standard rack...
  if StartingWeapon.HasKeyword(ArmorShield)
   ;Don't do anything
  else
  ; ...and doesn't have a dagger then place the weapon.
   PlayersDroppedWeapon = StartingWeapon
   HandleWeaponPlacement(TRUE)
  endif

 elseIf (RackType == 2)
  ; If this is a shield rack...
  if StartingWeapon.HasKeyword(ArmorShield)
   PlayersDroppedWeapon = StartingWeapon
   HandleWeaponPlacement(TRUE)
  else
```

```
  ;don't do anything
  endif

 elseIf (RackType == 3)
  ; If this is a COA weapon rack...
  if StartingWeapon.HasKeyword(WeaponTypeDagger)
   ;Don't do anything
  elseif StartingWeapon.HasKeyword(ArmorShield)
   ;Don't do anything
  elseif StartingWeapon.HasKeyword(WeaponTypeBow)
   ;Don't do anything
  else
  ; ...and doesn't have a dagger then place the weapon.
   PlayersDroppedWeapon = StartingWeapon
   HandleWeaponPlacement(TRUE)
  endif

 endif

EndFunction
```

This function basically takes the starting weapon, and if it is compatible with the RackType, assigns it to the property 'PlayersDroppedWeapon'. Essentially, it does the same thing as the first part of the 'HandleWeaponPlacement' function, except for the starting weapon instead of forcing the player to drop a weapon.

After assigning the variable, it will call on the 'HandleWeaponPlacement' function with the parameter set to true, so that it skips straight to the second half of the function. The only interesting thing to note for the 'HandleStartingWeapon' function is that it doesn't deal with RackType equal to 4. This means that you can't assign a starting weapon to a dagger display and have it work properly.

Conclusion

After going through this script, we have an understanding of what it's supposed to do. If it has a starting weapon, then it would hang the starting weapon onto its linked ref (the rack/plaque/display) when the cell is loaded. If the player activates it, it will cause the player to drop a weapon or a shield (depending on RackType) and then hang the weapon or shield onto the rack/plaque/display.

Putting It All Together

So when you go and use a display like a weapon plaque, you are actually activating the invisible activator in front of the plaque, not the plaque itself. When you do that, the activator will cause you to drop your weapon, and hang it onto its linked ref, the plaque.

When the plaque finds the weapon in its trigger area, it disables its own linked ref, the invisible activator. This means you can no longer activate it to hang on another weapon. If you now go and take back your weapon, the trigger area is clear and the plaque enables the invisible activator again. Now you can put a weapon on it again.

Because the trigger area is enabled and disabled by the rack script, it cannot also be subject to an enable parent. The racks that are built in Hearthfire homes do not have enable parents on the triggers for that reason. The racks themselves do have one.