Table of Contents

Introduction to Papyrus in Skyrim Modding

Papyrus is the scripting language used in Skyrim for creating custom content, enabling modders to extend the game's functionality in ways that go beyond the Creation Kit’s standard tools. From creating new spells to designing complex quests and customizing NPC behavior, Papyrus provides the flexibility and power needed to fully immerse players in your mods.

Mastering Papyrus allows you to control a wide range of game mechanics, such as managing inventory, responding to player actions, triggering events, and scripting advanced AI behavior. Understanding and utilizing Papyrus is a crucial step for any modder aiming to create dynamic and interactive experiences within Skyrim.

Before we dive into scripting, let’s get everything ready:

1. Install the Creation Kit

The **Creation Kit** is Bethesda’s official modding tool for Skyrim. You can download it via **Steam** under the "Tools" section.

- Once installed, it includes the **Papyrus script editor**, where you'll write and manage your scripts.

2. Prepare Skyrim and Your Mod

- Ensure you have a **clean installation** of Skyrim.
- Open the **Creation Kit** and create a new mod file (**.esp or .esm**) to store your scripts. This will serve as the foundation for all your modifications.

═══★═══════════★═══

First of all the basics:

═══★═══════════★═══

Enable Script Files:

Scripts need to be enabled within the Creation Kit. You can link scripts to various objects, quests, or NPCs by associating them with specific game elements.
Ensure you understand how to add your script to the object in question within the CK **interface.**

Basic Syntax and Structure

Papyrus is a relatively simple language to learn, but understanding the basic syntax and structure is crucial.

Statements:
Papyrus uses statements to perform actions. A statement is a single instruction, like setting a variable or calling a function.

```
int health = 100
string playerName = "Dragonborn"
```

Functions:
Functions are used to group a set of instructions that you can reuse. Functions in Papyrus are declared with the Function keyword.

```
Function HealPlayer()
    Game.GetPlayer().RestoreHealth(50)
EndFunction
```

Events:
Events are special functions that automatically trigger in response to certain game conditions, such as a player interacting with an object.

```
Event OnActivate(ObjectReference akActivator)
    Game.GetPlayer().RestoreHealth(50)
EndEvent
```

Blocks:
Code blocks are enclosed by If statements or functions and are used to group multiple actions together.

```
If Game.GetPlayer().GetHealth() < 50
    Game.GetPlayer().RestoreHealth(30)
EndIf
```

Variables and Data Types
In Papyrus, variables store data, and each variable must be defined with a specific data type. Common data types in Papyrus include:
Integer:
Used for whole numbers.

```
int health = 100
```

Float:
Used for numbers with decimals.

```
float distance = 5.5
```

```

```

String:
Used for text.

```
string playerName = "Dragonborn"
```

Actor:
Used for NPCs or creatures.

```
Actor Property MyNPC Auto
```

ObjectReference:
Used for references to objects in the game world.

```
ObjectReference Property MyDoor Auto
```

Functions and Events

Functions are blocks of reusable code, while events automatically trigger in response to certain game actions.

Function Declaration:
Functions allow you to perform specific actions when called.

```
Function HealPlayer()
    Game.GetPlayer().RestoreHealth(50)
EndFunction
```

Events:
Events are used to respond to game triggers, such as player actions.

```
Event OnActivate(ObjectReference akActivator)
    Game.GetPlayer().RestoreHealth(50)
EndEvent
```

Conditions and Control Flow
Papyrus supports various control structures to dictate the flow of your script based on conditions.

If Statement:

Executes code if a condition is true.

```
If Game.GetPlayer().GetHealth() < 50
```

```
    Game.GetPlayer().RestoreHealth(30)
EndIf
```

While Loop:
Repeats code as long as the condition is true.

```
While Game.GetPlayer().GetHealth() < 50
    Game.GetPlayer().RestoreHealth(10)
EndWhile
```

For Loop:
Repeats code a specific number of times.

```
For i = 0 To 10
    Debug.Trace("Loop iteration " + i)
EndFor
```

Working with Objects and Properties
Objects and properties are core to Skyrim modding. Objects refer to items, NPCs, and locations in the game world, while properties represent the specific attributes of those objects.

Creating a Reference

A reference to an object allows you to manipulate it within the game.

```
Actor Property MyNPC Auto
```

Accessing Object Properties:
You can change properties of an object, such as health, name, or position.

```
MyNPC.SetHealth(100
```

Creating and Using Scripts in the Creation Kit
To use scripts in the Creation Kit:

Create a Script:
In the CK, go to the Script Manager and create a new script. You'll be able to write Papyrus code here.

Assign the Script:
Once created, you can assign the script to an object (like an NPC or a door) in the CK interface.

Save and Compile:

After writing your script, save it and compile it. The Creation Kit will check for any errors or issues in your code.

Debugging and Optimization

Debugging is a crucial part of writing scripts in Papyrus. You can use Debug.Trace to log information to the Papyrus log file.

Using Debug.Trace:
It outputs messages that can help track the execution of your script.

```
Debug.Trace("Player health: " + Game.GetPlayer().GetHealth())
```

Optimization:
Avoid unnecessary loops.
Minimize event registration and deregistration.
Reduce the use of heavy operations in frequently called functions.

Practical Examples
Here are some practical examples to help you get started with Papyrus scripting:

Heal the Player on Activation:

```
Event OnActivate(ObjectReference akActivator)
    Game.GetPlayer().RestoreHealth(50)
EndEvent
```

Creating a Basic Quest Script:

```
Scriptname MyQuestScript extends Quest

Bool Property bIsQuestStarted Auto
Bool Property bIsQuestCompleted Auto

Function StartQuest()
    bIsQuestStarted = True
    Debug.Trace("Quest started!")
EndFunction

Function CompleteQuest()
    bIsQuestCompleted = True
    Debug.Trace("Quest completed!")
EndFunction
```

Custom Events
Custom events allow you to define your own triggers within the game.

Creating a Custom Event:
Define an event that gets called under specific conditions.

```
Event OnCustomEvent()
    Debug.Trace("Custom event triggered!")
EndEvent
```

Working with Forms and Aliases
Forms are used to reference objects like NPCs, items, and locations, while aliases represent references to these objects in your script.

Using Form Aliases:

```
Alias Property MyAlias Auto
```

Assigning Forms:

```
MyAlias.AssignReference(MyNPC)
```

Timers and Delayed Execution
Papyrus provides ways to execute code after a delay or at periodic intervals.

Setting a Timer:
Use a timer to delay the execution of a function.

```
float delayTime = 5.0
RegisterForSingleUpdate(delayTime)
```

Handling User Input
Papyrus can respond to user inputs, such as when the player interacts with an object or selects a dialogue option.

User Input Example:

```
Event OnActivate(ObjectReference akActivator)
    If akActivator == Game.GetPlayer()
        Debug.Trace("Player interacted with object")
    EndIf
EndEvent
```

Script Best Practices
To write efficient and clean code, follow these best practices:
Use Descriptive Variable Names: Make your code easier to read and maintain.
Comment Your Code: Always add comments explaining what each part of the code does.
Organize Your Scripts: Keep related code together and modularize it.

Advanced Topics
Now that you have the basics down, you can explore more advanced features:

Let's get started:

Registering for Events

Learn how to register your scripts to respond to in-game events like **combat, quest stages, item pickups, or NPC interactions**. Event registration is crucial for **reactive gameplay mechanics** and **dynamic interactions**.

═══★═══════════★═══

1. Understanding Event Registration in Papyrus
Papyrus uses **Event-Driven Programming**, where scripts can **listen for specific in-game actions** and execute code in response.
✅ **OnCombatStateChanged** – Detects when an actor enters or leaves combat.
✅ **OnLocationChange** – Detects when the player moves between locations.
✅ **OnObjectEquipped** – Detects when an item is equipped.
✅ **OnPlayerLoadGame** – Detects when the player loads a save.
✅ **OnQuestStageSet** – Triggers actions when a quest stage is updated.

You **must register for some events manually** using `RegisterForSingleUpdate()` or `RegisterForModEvent()`

2. Listening for Combat Events

A. Detecting When an NPC Enters Combat

```
Event OnCombatStateChanged(Actor akTarget, int aeCombatState)
If aeCombatState == 1 ; 1 = Entering Combat
Debug.Notification(akTarget.GetDisplayName() + " has entered
combat!")
ElseIf aeCombatState == 0 ; 0 = Leaving Combat
Debug.Notification(akTarget.GetDisplayName() + " has left
combat.")
EndIf
EndEvent
```

📝 **Explanation:**

- Displays a notification when an **NPC enters or exits combat**.
- Can be used to **trigger combat AI adjustments**.

3. Registering for Quest Stage Events

A. Triggering an Effect at a Specific Quest Stage

```
Quest Property MyQuest Auto

Event OnQuestStageSet(int aiStageID, int aiNewStage)
If aiStageID == 30
Debug.Notification("The quest has reached stage 30!")
Game.GetPlayer().AddItem(Gold001, 100) ; Reward the player
EndIf
EndEvent
```

📝 **Explanation:**

- When the **quest reaches stage 30**, the **player receives 100 gold**.
- Useful for **scripted rewards or story progression**.

═══★═══════════★═══

4. Registering for Player Actions

A. Detecting When the Player Equips an Item

```
Event OnObjectEquipped(Form akBaseObject, ObjectReference
akReference)
If akBaseObject == MySpecialSword
Debug.Notification("You equipped the legendary sword!")
EndIf
EndEvent
```

📝 **Explanation:**

- Detects when a specific weapon is equipped.
- Useful for custom gear mechanics.

5. Registering for Location Changes

A. Detecting When the Player Enters a New Location

```
Event OnLocationChange(Location akOldLoc, Location akNewLoc)
Debug.Notification("You have entered " + akNewLoc.GetName() + "!")
EndEvent
```

📝 **Explanation:**

- Triggers a message when the player changes locations.
- Can be used for ambushes, dynamic music, or weather changes.

═══★═══════════★═══

6. Using Custom Events with ModEvent

Papyrus allows you to create your own events with `ModEvent`

A. Creating and Sending a Custom Event

Step 1: Sending the Event

```
int MyEvent = ModEvent.Create("CustomEvent")
ModEvent.PushForm(MyEvent, Game.GetPlayer())
ModEvent.Send(MyEvent)
```

Step 2: Registering and Handling the Event

```
Event OnInit()
RegisterForModEvent("CustomEvent", "OnCustomEventReceived")
EndEvent

Event OnCustomEventReceived(Form akSender)
Debug.Notification(akSender.GetDisplayName() + " triggered the
custom event!")
EndEvent
```

📝 **Explanation:**

- Creates a new custom event and sends it.
- The script listens for the event and responds.
- Useful for mod communication and advanced mechanics.

7. Timed Events with RegisterForSingleUpdate()

A. Running a Function Every Few Seconds

```
Event OnInit()
RegisterForSingleUpdate(5.0) ; Calls OnUpdate() after 5 seconds
EndEvent

Event OnUpdate()
Debug.Notification("5 seconds have passed!")
RegisterForSingleUpdate(5.0) ; Re-register for another update
EndEvent
```

📝 **Explanation:**

- Runs a function every 5 seconds.
- Useful for timed events, weather changes, or periodic updates.

═══★═══════════★═══

8. Unregistering Events to Prevent Save Bloat

When using event registrations, always unregister them when no longer needed to avoid memory leaks.

A. Unregistering a Single Update Event

```
UnregisterForUpdate()
```

B. Unregistering Mod Events

```
UnregisterForModEvent("CustomEvent")
```

📝 **Explanation:**

- Prevents unnecessary script executions, improving performance

Animation Handling

Control animations for NPCs or objects to enhance immersion by utilizing **Papyrus scripting** and **animation frameworks** like **FNIS**, **Nemesis**, and **Dynamic Animation Replacer (DAR)**.

1. Understanding Skyrim's Animation System

Skyrim's animations are controlled using **Behavior Graphs**, **Animation Events**, and **Animation Modifiers**. With Papyrus, you can:
✅ **Trigger animations dynamically** based on in-game events.
✅ **Override NPC animations** to create custom behaviors.
✅ **Play specific animations for objects, creatures, or the player.**
✅ **Combine animations with AI packages** for advanced interactions.

═══★═══════════★═══

2. Playing an Animation via Papyrus

You can play specific animations using the `PlayIdle()` or `SendAnimationEvent()` functions.

A. Playing an Idle Animation on an NPC

```
Idle Property SitIdle Auto
Actor Property TargetNPC Auto

Event OnActivate(ObjectReference akActivator)
TargetNPC.PlayIdle(SitIdle)
EndEvent
```

📝 **Explanation:**

- When the player **activates an object**, the **NPC plays the sitting animation**.
- Useful for **interactive NPC behaviors**.

B. Triggering an Animation on the Player

```
Idle Property PlayerBow Auto

Event OnHit(ObjectReference akAggressor, Form akSource, Projectile
akProjectile, bool abPowerAttack, bool abSneakAttack, bool
abBashAttack, bool abHitBlocked)
Game.GetPlayer().PlayIdle(PlayerBow)
Debug.Notification("You bow in respect!")
EndEvent
```

📝 **Explanation:**

- When the player **is hit**, they **play a bow animation**.
- This can be used for **roleplaying interactions**.

═══★═══════════★═══

3. Using Animation Events for More Control

Animation events allow you to trigger effects when an animation reaches a specific point.

A. Detecting When an NPC Draws Their Weapon

```
Event OnAnimationEvent(ObjectReference akSource, string
asEventName)
If asEventName == "WeaponDraw"
Debug.Notification(akSource.GetDisplayName() + " has drawn their
weapon!")
EndIf
EndEvent
```

📝 **Explanation:**

- Detects when an **NPC draws their weapon** and displays a message.
- Can be used to trigger **combat mechanics**.

4. Advanced Animation Swaps with DAR and OAR

Dynamic Animation Replacer (DAR) and **Open Animation Replacer (OAR)** allow you to swap animations without modifying behavior files.

A. Applying Custom Animations to a Specific NPC

```
Event OnInit()
If TargetNPC.GetActorBase().GetRace() == ArgonianRace
TargetNPC.SendAnimationEvent("SpecialArgonianIdle")
EndIf
EndEvent
```

📝 **Explanation:**

- Applies **a unique idle animation to Argonians**.
- Useful for **race-based or faction-based animation changes**.

═══★═══════════★═══

5. Object Animations and Physics Interactions

You can animate objects such as **doors, levers, or moving platforms**.

A. Moving a Door with Animation Events

```
ObjectReference Property MyDoor Auto

Event OnTriggerEnter(ObjectReference akActionRef)
If akActionRef == Game.GetPlayer()
MyDoor.PlayAnimation("Open")
EndIf
EndEvent
```

📝 **Explanation:**

- The **door opens when the player enters a trigger zone**.
- Can be used for **traps, puzzles, or interactive doors**.

6. Combining Animations with AI Packages

AI packages can make NPCs **move, interact, or perform scripted animations**.

A. Making an NPC Perform a Custom Animation on a Schedule

```
Package Property CustomDancePackage Auto
Actor Property TargetNPC Auto

Event OnInit()
TargetNPC.SetPackageOverride(CustomDancePackage)
TargetNPC.EvaluatePackage()
EndEvent
```

📝 **Explanation:**

- Forces the NPC to **use a dancing package**.
- Useful for **scripted events or immersive interactions**.

═══★═══════════★═══

7. Dynamic Combat Animations

You can modify combat animations based on conditions like stamina or skill level.

A. Changing Attack Animations at Low Stamina

```
Event OnHit(ObjectReference akAggressor, Form akSource, Projectile
akProjectile, bool abPowerAttack, bool abSneakAttack, bool
abBashAttack, bool abHitBlocked)
If Game.GetPlayer().GetActorValue("Stamina") < 20
Game.GetPlayer().SendAnimationEvent("TiredAttack")
Debug.Notification("You are too tired to swing properly!")
EndIf
EndEvent
```

📝 **Explanation:**

- When stamina is low, the player uses a slower attack animation.
- This allows for realistic fatigue-based combat.

8. Custom Animation Triggers for Quests and Events

Animations can be used to create cinematic moments in quests.

A. Playing an Animation During a Quest Scene

```
Scene Property MyScene Auto
Actor Property QuestNPC Auto
Idle Property AngryGesture Auto

Event OnQuestStageSet(int aiStageID, int aiNewStage)
If aiStageID == 20
QuestNPC.PlayIdle(AngryGesture)
Debug.Notification("The NPC is furious!")
EndIf
EndEvent
```

📝 **Explanation:**

- At quest stage 20, the NPC plays an angry gesture.
- Can be used for scripted story moments

═══★═══════════★═══

Working with Perks and Abilities

Write scripts that **interact with Skyrim's perk and ability systems**, allowing you to **add, modify, or remove perks and abilities dynamically**.

═══★═══════════★═══

1. Introduction to Perks and Abilities

Perks and abilities are fundamental to Skyrim's **gameplay mechanics**. Through **Papyrus scripting**, you can:

✅ **Add or remove perks dynamically** based on player choices.
✅ **Modify existing perks** to fine-tune their effects.
✅ **Grant abilities and passive effects** via scripting.
✅ **Create new perk-based mechanics** that react to gameplay events.

Perks belong to **Skill Trees**, while **Abilities** are passive effects applied to actors.

2. Adding and Removing Perks via Script

Perks can be added to or removed from an actor using **AddPerk()** and **RemovePerk()**.

A. Granting a Perk on Level Up

```
Perk Property MyCustomPerk Auto

Event OnLevelUp()
If Game.GetPlayer().GetLevel() == 10
Game.GetPlayer().AddPerk(MyCustomPerk)
Debug.Notification("You have unlocked a new perk!")
EndIf
EndEvent
```

📝 **Explanation:**

- When the **player reaches level 10**, they **gain a new perk automatically**.
- This is useful for **custom progression systems**.

═══★═══════════════★═══

B. Removing a Perk After a Certain Event

```
Event OnItemAdded(Form akBaseItem, int aiItemCount,
ObjectReference akItemReference, ObjectReference
akSourceContainer)
If akBaseItem == CursedArtifact
Game.GetPlayer().RemovePerk(MyCustomPerk)
Debug.Notification("The cursed artifact has weakened you!")
EndIf
EndEvent
```

📝 **Explanation:**

- If the player **picks up a cursed artifact**, they **lose a perk**.
- This can be used for **unique debuffs or consequences**.

3. Creating Custom Perk-Based Effects

Some abilities need **custom scripting** beyond the Creation Kit's built-in perk effects.

A. Checking If the Player Has a Specific Perk

```
If Game.GetPlayer().HasPerk(MyCustomPerk)
Debug.Notification("Your special skill activates!")
EndIf
```

📝 **Explanation:**

- This can be used to create **perk-based reactions**, such as **unlocking dialogue options** or **activating special attacks**.

═══★═══════════════★═══

4. Granting Abilities to Actors

Abilities are passive effects added via **Spell Effects (Magic Effects)**.

A. Adding a Passive Ability via Script

```
Spell Property MyAbility Auto

Event OnInit()
Game.GetPlayer().AddSpell(MyAbility, False)
Debug.Notification("You have gained a new passive ability!")
EndEvent
```

📝 **Explanation:**

- The **player gains a new ability** when the script initializes.
- This is useful for **custom passive effects** like regeneration or resistance.

B. Removing an Ability Under Specific Conditions

```
Event OnEnterBleedout()
Game.GetPlayer().RemoveSpell(MyAbility)
Debug.Notification("Your power fades as you fall unconscious!")
EndEvent
```

📝 **Explanation:**

- If the **player enters bleedout state (near death)**, they **lose an ability**.
- This can be used for **realistic injury mechanics**.

═══★═══════════════★═══

5. Creating Dynamic Perk-Based Mechanics

Perks can be used to trigger **unique gameplay mechanics**.

A. Increasing Weapon Damage with a Custom Perk

```
Event OnHit(ObjectReference akAggressor, Form akSource, Projectile
akProjectile, bool abPowerAttack, bool abSneakAttack, bool
abBashAttack, bool abHitBlocked)
If Game.GetPlayer().HasPerk(MyDamageBoostPerk)
akAggressor.DamageActorValue("Health", 20) ; Extra damage
Debug.Notification("Your perk grants bonus damage!")
EndIf
EndEvent
```

📝 **Explanation:**

- If the **player has a specific perk**, their **attacks deal extra damage**.
- This allows **custom combat mechanics** beyond vanilla perks.

6. Implementing Perk-Based Interactions

You can use perks to unlock special dialogue, items, or abilities.

A. Unlocking a Secret Dialogue Option with a Perk

```
If Game.GetPlayer().HasPerk(MySpeechPerk)
DialogueOption.Show()
EndIf
```

📝 **Explanation:**

- If the player has a speech perk, a secret dialogue option appears.
- This adds roleplaying depth to the game.

═══★═══════════════★═══

7. Creating a Perk Tree Progression System

You can script a custom perk system outside of Skyrim's standard skill trees.

A. Unlocking a New Perk Based on Experience Points

```
GlobalVariable Property ExperiencePoints Auto
Perk Property AdvancedCombatPerk Auto

Event OnUpdate()
If ExperiencePoints.GetValue() >= 100
Game.GetPlayer().AddPerk(AdvancedCombatPerk)
Debug.Notification("You have mastered advanced combat!")
EndIf
RegisterForSingleUpdate(5)
EndEvent
```

📝 **Explanation:**

- When the player reaches 100 experience points, they unlock a perk.
- This can be used for RPG-style progression systems.

8. Perk Synergy Mechanics

Perks can interact with each other to create synergies.

A. Combining Perks for a Unique Effect

```
If Game.GetPlayer().HasPerk(FireMastery) &&
Game.GetPlayer().HasPerk(ExplosivePower)
Game.GetPlayer().AddSpell(FireExplosionAbility, False)
Debug.Notification("Your fire spells now cause explosions!")
EndIf
```

📝 **Explanation:**

- If the player has both Fire Mastery and Explosive Power, they unlock a new ability.
- This creates interesting character builds

═══★═══════════★═══

Creating Dynamic NPC Behaviors

Create **complex behaviors** for NPCs based on **game events, player actions**, and environmental conditions.

═══★═══════════★═══

1. Introduction to Dynamic NPC Behaviors

In vanilla Skyrim, NPCs follow **predefined schedules** using AI Packages. However, through **Papyrus scripting**, we can create **dynamic behaviors**, such as:

✅ **NPCs reacting to player actions** (e.g., fleeing when low on health).
✅ **Custom AI routines** (e.g., patrols, schedules, and combat strategies).
✅ **Dynamic personality shifts** (e.g., becoming hostile based on faction changes).
✅ **Event-driven responses** (e.g., NPCs reacting to nearby battles).

These behaviors can make Skyrim's world **feel more alive and immersive**.

2. Using AI Packages for NPC Behavior

AI Packages define what an NPC does in specific conditions. They can be modified dynamically using **Papyrus**.

A. Assigning an AI Package to an NPC
To assign a **custom AI package** to an NPC, use the `SetPackage()` function.

```
Actor Property MyNPC AutoPackage Property NewPackage Auto

Event OnTriggerEnter(ObjectReference akActivator)
If akActivator == Game.GetPlayer()
MyNPC.SetPackage(NewPackage)
Debug.Notification("NPC behavior changed!")
EndIf
EndEvent
```

📝 **Explanation:**

- When the **player enters a trigger zone**, the NPC **switches to a new AI package**.
- This can be used for **custom patrols, scripted attacks, or defensive behaviors**.

═══★═══════════════★═══

3. Reacting to Player Actions

NPCs can dynamically change behavior based on **player choices, crimes, or dialogue**.

A. Making an NPC Flee When Low on Health

```
Event OnHit(ObjectReference akAggressor, Form akSource, Projectile
akProjectile, bool abPowerAttack, bool abSneakAttack, bool
abBashAttack, bool abHitBlocked)
If Self.GetActorValue("Health") < 25
Self.SetPackage(FleePackage)
Debug.Notification("NPC is fleeing!")
EndIf
EndEvent
```

📝 **Explanation:**

- When the NPC's **health drops below 25**, they **switch to a fleeing package**.
- This can be useful for **cowardly NPCs or tactical retreats**.

B. Changing NPC Aggression Based on Player Choices

```
Event OnItemAdded(Form akBaseItem, int aiItemCount,
ObjectReference akItemReference, ObjectReference
akSourceContainer)
If akBaseItem == StolenItem
Self.SetRelationshipRank(Game.GetPlayer(), -4)
Self.StartCombat(Game.GetPlayer())
Debug.Notification("The merchant caught you stealing!")
EndIf
EndEvent
```

📝 **Explanation:**

- If the player **steals from an NPC**, their **relationship rank decreases** and they **attack the player**.
- This creates a **reactive AI system**, where NPCs enforce **custom laws or morality**.

═══★═══════════★═══

4. Implementing NPC Decision-Making

NPCs can be given logic-based decision-making based on environmental conditions.

A. Making NPCs Take Cover in Combat

```
Event OnCombatStateChanged(Actor akTarget, int aeCombatState)
If aeCombatState == 1 ; NPC enters combat
If Self.GetActorValue("Health") < 50
Self.SetPackage(CoverPackage)
Debug.Notification("NPC is taking cover!")
EndIf
EndIf
EndEvent
```

📝 **Explanation:**

- If the NPC's health falls below 50%, they find cover instead of charging forward.
- This adds realism and tactical behavior.

5. Creating Patrols and Guard Routines

NPCs can be given dynamic patrol paths that change based on the time of day or external events.

A. Making Guards Switch Between Day and Night Shifts

```
Event OnUpdate()
If Utility.GetCurrentGameTime() > 18 ||
Utility.GetCurrentGameTime() < 6
Self.SetPackage(NightPatrol)
Else
Self.SetPackage(DayPatrol)
EndIf
RegisterForSingleUpdate(1)
EndEvent
```

📝 **Explanation:**

- NPCs switch between patrol routes based on the in-game time of day.
- This is useful for realistic guard shifts or dynamic patrols.

═══★═══════════════★═══

6. NPC Reactions to Nearby Events

NPCs can respond dynamically to nearby battles, weather changes, or explosions.

A. Making NPCs React to Nearby Battles

```
Event OnCombatStateChanged(Actor akTarget, int aeCombatState)
If aeCombatState == 1 ; NPC enters combat
If Game.GetPlayer().GetDistance(Self) < 500
Self.SetPackage(AssistPlayerPackage)
Debug.Notification("The NPC is helping you!")
EndIf
EndIf
EndEvent
```

📝 **Explanation:**

- If the player is near an NPC during combat, the NPC will assist them.
- This is useful for ally mechanics or faction-based reinforcements.

7. Implementing Randomized NPC Behavior

Adding random elements to NPC behavior makes the game world feel less predictable.

A. Giving NPCs Random Idle Actions

```
Event OnUpdate()
int action = Utility.RandomInt(0, 2)

If action == 0
Self.PlayIdle(Idle_Sit)
ElseIf action == 1
Self.PlayIdle(Idle_Stretch)
Else
Self.PlayIdle(Idle_LookAround)
EndIf

RegisterForSingleUpdate(10)
EndEvent
```

📝 **Explanation:**

- Every 10 seconds, the NPC performs a random idle animation.
- This prevents NPCs from standing still unnaturally.

═══★═══════════★═══

8. Making NPCs Comment on the Player's Actions

NPCs can dynamically react to player choices, adding more depth to roleplaying.

A. NPCs Commenting on the Player's Armor

```
Event OnPlayerLoadGame()
If Game.GetPlayer().IsEquipped(DragonArmor)
Self.Say(CustomVoiceLine, True, None)
EndIf
EndEvent
```

📝 **Explanation:**

- If the player equips dragon armor, NPCs comment on it dynamically.
- This makes player choices feel more meaningful

Implementing MCM (Mod Configuration Menu)

Learn how to create a **customizable menu** for your mod using **Mod Configuration Menu (MCM)**, allowing players to adjust mod settings directly in-game.

═══★═══════════════★═══

1. What is MCM?

The **Mod Configuration Menu (MCM)** is a **SkyUI-based** menu system that allows modders to provide:
✅ **User-friendly UI** for adjusting mod settings.
✅ **Sliders, checkboxes, dropdowns**, and other interactive elements.
✅ **Persistent settings** stored across game sessions.
✅ **Integration with SKSE** for enhanced functionality.
MCM is **required for many popular mods**, including **Frostfall, Campfire, iNeed, and Wildcat**

2. Setting Up MCM for Your Mod

Before creating an MCM menu, ensure that:
✅ You have **SkyUI installed** (it includes the MCM framework).
✅ Your mod **uses SKSE** (for persistent settings).
✅ You understand **basic Papyrus scripting**.
To use MCM, create a **new script** extending `MCM_ConfigBase`
═══★═══════════════★═══

3. Creating a Basic MCM Menu

A. Creating the Script File

Create a new **Papyrus script** in the Creation Kit or an external editor like Sublime Text.

```
Scriptname MyMod_MCM extends SKI_ConfigBase

Event OnConfigInit()
SetName("My Mod Settings") ; Set MCM menu name
SetVersion(1)
EndEvent
```

📝 **Explanation:**

- `SetName("My Mod Settings")` defines the menu title.
- `SetVersion(1)` sets the version for tracking updates.

4. Adding Custom Settings

MCM supports multiple setting types: **sliders, checkboxes, dropdowns, and text inputs**.

A. Adding a Toggle (Checkbox) Setting

```
Bool Property EnableFeature Auto

Event OnPageReset()
AddToggleOptionST("EnableFeature", "Enable Special Feature",
EnableFeature)
EndEvent

Event OnOptionSelectST(String optionName, Bool newValue)
If optionName == "EnableFeature"
EnableFeature = newValue
Debug.Notification("Feature is now: " + EnableFeature)
EndIf
EndEvent
```

📝 **Explanation:**

- `AddToggleOptionST()` creates a **checkbox** in MCM.
- `OnOptionSelectST()` handles **player input**, updating the feature.

═══★═══════════★═══

B. Adding a Slider for Custom Values

```
Float Property DifficultyMultiplier Auto

Event OnPageReset()
AddSliderOptionST("DifficultyMultiplier", "Set Difficulty",
DifficultyMultiplier, 0.5, 3.0, 0.1)
EndEvent

Event OnOptionSliderAcceptST(String optionName, Float newValue)
If optionName == "DifficultyMultiplier"
DifficultyMultiplier = newValue
Debug.Notification("Difficulty set to: " + DifficultyMultiplier)
EndIf
EndEvent
```

📝 **Explanation:**

- `AddSliderOptionST()` creates a **slider** with **min, max, and step values**.
- `OnOptionSliderAcceptST()` saves the selected **difficulty level**.

C. Adding a Dropdown Menu

```
Int Property SelectedMode

AutoEvent OnPageReset()
String[] options = new String[3]
options[0] = "Easy"
options[1] = "Normal"
options[2] = "Hard"
AddMenuOptionST("SelectedMode", "Select Mode", SelectedMode,
options)
EndEvent

Event OnOptionMenuSelectST(String optionName, Int newIndex)
If optionName == "SelectedMode"
SelectedMode = newIndex
Debug.Notification("Selected Mode: " + SelectedMode)
EndIf
EndEvent
```

📝 **Explanation:**

- `AddMenuOptionST()` creates a **dropdown menu** with selectable values.
- `OnOptionMenuSelectST()` updates the chosen setting.

═══★═══════════★═══

5. Storing and Loading Settings with SKSE

MCM settings should be **saved and loaded automatically** to persist across sessions.

A. Saving Settings on Exit

```
Event OnConfigClose()
ModSettingSave()
EndEvent
```

B. Loading Settings on Startup

```
Event OnGameLoad()
ModSettingLoad()
EndEvent
```

📝 **Explanation:**

- `ModSettingSave()` ensures settings **persist** when closing MCM.
- `ModSettingLoad()` restores saved values on **game start**.

═══★═══════════★═══

6. Organizing MCM with Multiple Pages

For larger mods, split settings into **multiple pages**.

```
Event OnConfigInit()
AddPage("General")
AddPage("Advanced")
EndEvent

Event OnPageReset()
If CurrentPage == 0 ; General Settings
AddToggleOptionST("EnableFeature", "Enable Feature",
EnableFeature)
ElseIf CurrentPage == 1 ; Advanced Settings
AddSliderOptionST("DifficultyMultiplier", "Set Difficulty",
DifficultyMultiplier, 0.5, 3.0, 0.1)
EndIf
EndEvent
```

📝 **Explanation:**

- `AddPage("General")` and `AddPage("Advanced")` create **tabbed menus**.
- `CurrentPage` tracks the **active page**.

═══★═══════════★═══

7. Debugging and Testing MCM Menus

To test your MCM menu in-game:

1. Install the mod and **ensure SkyUI is active**.
2. Open the **Mod Configuration Menu** in Skyrim.
3. Navigate to your **mod's menu** and test all settings.
4. Check for errors with:`showlog`
5. If settings don't save, verify `ModSettingSave()` and `ModSettingLoad()` are called correctly.

8. Best Practices for MCM Scripting

✅ **Keep menus simple** – avoid excessive options in one page.
✅ **Use clear labels** – players should understand settings easily.
✅ **Ensure values are within limits** – prevent extreme values from breaking gameplay.
✅ **Test across multiple save files** – verify settings persist correctly.
✅ **Optimize script performance** – minimize unnecessary calculations in `OnOptionSelectST()`

═══★═══════════★═══

Writing SKSE-Enhanced Scripts

Extend your scripts with the **Skyrim Script Extender (SKSE)** to access additional functionality and create more advanced gameplay mechanics. SKSE expands Papyrus scripting, allowing access to additional **native functions**, **custom events**, and **performance improvements** that are not possible with vanilla scripting alone.

═══★═══════════★═══

1. What is SKSE?

The **Skyrim Script Extender (SKSE)** is a **modding tool** that enhances Skyrim's scripting capabilities. It allows modders to:
✅ Use **advanced scripting functions** beyond the limitations of vanilla Papyrus.
✅ Access **new gameplay mechanics**, UI modifications, and complex interactions.
✅ Improve **performance and efficiency** by reducing script load times.
✅ Create **custom events and properties** to expand modding possibilities.

SKSE is required for **many advanced mods**, including **RaceMenu, SkyUI, FNIS, Nemesis, and PapyrusUtil**.

═══★═══════════★═══

2. Setting Up SKSE for Papyrus Scripting

Before writing SKSE-based scripts, ensure that:
✅ You have **SKSE installed** (matching your Skyrim AE/SE version).
✅ The **SKSE Papyrus functions** are properly loaded.
✅ You are using a **script editor** like **Creation Kit or Sublime Text with Papyrus Syntax**.

To access SKSE functions in your script, include the SKSE library:

```
Scriptname MySKSEScript extends Quest
Import SKSE ; Enables access to SKSE functions
```

═══★═══════════★═══

3. Using SKSE Functions in Scripts

SKSE adds **hundreds of new functions** that make scripting more powerful and efficient. Here are some examples:

A. Getting a Player's Memory Usage
Monitor **how much memory the player is using**, which is useful for preventing crashes due to memory overload.

```
Int memoryUsed = Memory.GetVMSize()
Debug.Notification("Current memory usage: " + memoryUsed)
```

📝 **Explanation:**

- `Memory.GetVMSize()` retrieves the current **virtual memory** usage of the Skyrim script engine.
- Useful for **debugging and performance monitoring** in mods.

═══★═══════════★═══

B. Checking if a Key is Pressed
With SKSE, you can detect **keyboard inputs** directly in Papyrus.

```
If Input.IsKeyPressed(57) ; Spacebar (default Jump)
Debug.Notification("Jump key is pressed!")
EndIf
```

📝 **Explanation:**

- `Input.IsKeyPressed(KeyCode)` checks if a key is currently being held down.
- **Vanilla Papyrus lacks real-time key detection**, making this function **essential for custom controls**

C. Storing Persistent Data with PapyrusUtil

PapyrusUtil (an SKSE plugin) allows **saving persistent data** without relying on game save files.

```
String key = "PlayerHealth"
Float playerHealth = 100.0

StorageUtil.SetFloatValue(Game.GetPlayer(), key, playerHealth)
Float storedHealth = StorageUtil.GetFloatValue(Game.GetPlayer(),
key)

Debug.Notification("Stored Health: " + storedHealth)
```

📝 **Explanation:**

- `StorageUtil.SetFloatValue()` stores custom variables (health, stats, settings) without modifying save files.
- Unlike global variables, this prevents save bloat.

═══★═══════════★═══

4. Creating Custom Events with SKSE

SKSE allows you to create custom events, making scripts more modular and reusable.

A. Defining a Custom Event

Create a new event that can be triggered from other scripts.

```
Scriptname MyCustomEventScript extends Quest

Event OnPlayerJump()
Debug.Notification("Player has jumped!")
EndEvent
```

📝 **Explanation:**

- `OnPlayerJump()` is a custom event that can be called whenever needed.

B. Sending a Custom Event from Another Script

Call your event from another script.

```
MyCustomEventScript.SendCustomEvent("OnPlayerJump")
```

📝 **Explanation:**

- `SendCustomEvent("EventName")` triggers the custom event.
- Useful for complex interactions where multiple scripts need to communicate.

═══★═══════════★═══

5. Accessing UI Elements with SKSE (SkyUI Lib)

With SkyUI Lib (an SKSE extension), you can interact with custom UI elements.

A. Displaying a Custom Message Box

```
Int buttonPressed = UIExtensions.ShowMessageBox("Do you accept the
quest?")
If buttonPressed == 0
Debug.Notification("Quest accepted!")
Else
Debug.Notification("Quest declined.")
EndIf
```

📝 **Explanation:**

- This function creates a pop-up message with Yes/No options.
- Great for quest mods and user interactions.

═══★═══════════★═══

6. Improving Performance with SKSE

SKSE allows optimized scripting, reducing lag and script load times.

A. Running a Function in a Background Thread

```
RegisterForSingleUpdate(1.0) ; Delays script execution
```

📝 **Explanation:**

- Delays script execution to prevent Skyrim's scripting engine from overloading.
- Reduces script lag in complex mods.

7. Debugging and Profiling SKSE Scripts

SKSE offers powerful debugging tools.

A. Checking Script Execution Time

```
Float startTime = Utility.GetCurrentRealTime(); Execute script logic
Float endTime = Utility.GetCurrentRealTime()
Debug.Trace("Script executed in " + (endTime - startTime) + " seconds")
```

📝 **Explanation:**

- Measures how long a script takes to run.
- Helps optimize performance by identifying slow code.

═══★═══════════★═══

8. Best Practices for Writing SKSE Scripts

✅ Use SKSE functions only when necessary – don't overuse them for simple tasks.
✅ Keep scripts lightweight – avoid unnecessary memory usage.
✅ Always check for SKSE version compatibility – outdated functions can cause crashes.
✅ Test scripts thoroughly – SKSE functions may behave differently on various game versions.
✅ Use `Debug.Trace()` for logging – track errors and debug scripts efficiently.

═══★═══════════★═══

Managing Save Bloat and Memory Usage

When creating mods for Skyrim, one critical factor to consider is save bloat and memory usage. These two issues can cause significant performance degradation over time, leading to slow load times, larger save file sizes, and potential crashes. By managing your script's memory usage carefully, you can avoid these problems and ensure your mod runs efficiently without harming the player's experience.

1. What is Save Bloat?

Save bloat occurs when save files grow excessively in size due to accumulated data. This can happen due to:
✅ Persistent variables in scripts that remain active even after the player exits the game.
✅ Scripted objects that aren't properly cleaned up or removed.
✅ Excessive use of `Global` variables which store information indefinitely.
As a result, save files become larger and more cumbersome, making loading times slower and potentially leading to performance drops or crashes.

═══★═══════════★═══

2. Memory Leaks in Skyrim

A memory leak occurs when a script allocates memory but fails to release it when it's no longer needed. Over time, this increases the amount of allocated memory without freeing up space, causing Skyrim to run out of memory, eventually leading to crashes or lag.
Common causes of memory leaks include:

✅ Persistent references that are not properly removed after they're no longer in use.
✅ Overusing `Global` variables without cleaning up or resetting them.
✅ Objects or actors that are left lingering in memory even when they're no longer needed.

═══★═══════════★═══

3. Best Practices to Avoid Save Bloat and Memory Leaks

A. Properly Handle Persistent References
In Skyrim, persistent references (e.g., objects, NPCs, or items) that are not cleaned up properly can accumulate in save files, leading to save bloat. To prevent this, always ensure that you delete or reset persistent references when they're no longer needed.

Example 1: Removing Persistent References

```
Scriptname RemovePersistentActor extends Actor

Actor targetActor

; Initialize the script with the actor to remove
Event OnInit()
targetActor = Game.GetPlayer().GetFollower()
EndEvent

Function Cleanup()
If targetActor.IsDead()
targetActor.Delete() ; Removes the actor from memory
targetActor = None
EndIf
EndFunction
```

📝 **Explanation:**

- This script will remove an actor from memory when the actor is dead and no longer needed.
- Using `Delete()` ensures the actor is properly removed from memory

═══★═══════════★═══

B. Avoid Global Variables for Temporary Data

Global variables are useful but can lead to save bloat if used for temporary data. Instead, use local variables or quests that don't persist through saves.

Example 2: Avoiding Globals for Temporary Data

```
Scriptname QuestSaveData extends Quest

; Using local variables instead of globals
Function SaveQuestData()
String questStatus = "Completed"
Int questID = 1001
Debug.Notification("Quest " + questID + ": " + questStatus)
EndFunction
```

📝 **Explanation:**

- In this case, the script uses local variables `questStatus` and `questID` instead of storing data in a global variable that persists across saves.

C. Cleanup After Scripts Are Done
When a script finishes running or an event ends, always make sure to clean up any resources or references the script was using. This ensures there are no lingering objects consuming memory.

Example 3: Cleaning Up After an Event

```
Scriptname CleanUpAfterEvent extends Quest

ObjectReference objRef

; Cleanup function to remove unused references
Function Cleanup()
If objRef
objRef.Disable() ; Disable the object if it's no longer needed
objRef = None
EndIf
EndFunction

; Ensure cleanup occurs at the end of a quest
Event OnQuestComplete()
Cleanup() ; Call cleanup when quest ends
EndEvent
```

📝 **Explanation:**

- The script disables the object reference and nullifies the variable once it is no longer needed, ensuring it doesn't consume unnecessary memory.

D. Use Object Pools for Frequently Used Objects
If your mod involves creating and destroying objects frequently (e.g., projectiles, NPCs), consider using an object pool to recycle objects instead of creating and deleting them constantly. This prevents the game from having to repeatedly allocate and deallocate memory.

Example 4: Object Pool Implementation

```
Scriptname ObjectPoolManager extends Quest

ObjectReference pooledObjects[10]

Function GetObjectFromPool()
; Reuse objects from the pool to avoid constant
creation/destruction
For i = 0 to 9
If pooledObjects[i] == None
pooledObjects[i] = Game.GetPlayer().PlaceAtMe(FormID) ; Place a
new object
Return pooledObjects[i]
EndIf
EndFor
Return None
EndFunction

Function ReturnObjectToPool(ObjectReference obj)
; Return the object to the pool for reuse
For i = 0 to 9
If pooledObjects[i] == obj
pooledObjects[i] = None
obj.Disable()
EndIf
EndFor
EndFunction
```

📝 **Explanation:**

- The script reuses objects from a predefined pool rather than constantly creating new ones. This reduces memory allocation and improves performance.

4. Properly Manage Scripted Actors and Quests

When working with scripted actors or quest-related objects, be sure to clean up after them to prevent them from becoming persistent when they are no longer relevant.

Example 5: Removing an Actor After Quest Completion

```
Scriptname QuestActorCleanup extends Quest

Actor questActor

; Cleanup when the quest is completed
Event OnQuestComplete()
questActor.Delete() ; Remove the actor from the world
questActor = None ; Nullify the reference
EndEvent
```

📝 **Explanation:**

- Once the quest is completed, the actor is removed from the world and the reference is nullified, avoiding memory buildup.

═══★═══════★═══

5. Avoiding Save Bloat by Managing Large Datasets

If your mod involves storing large amounts of data (e.g., inventories, large world data), be mindful of how you store and handle this data.

Best Practices for Large Datasets

- Use **JSON** or **XML files** for storing large amounts of data, allowing it to be loaded only when needed.
- Avoid storing large arrays or lists in memory permanently. Instead, load them dynamically during gameplay and unload when not needed.
- Consider using batch processing to load and unload data to keep memory usage efficient.

6. Memory Management Tips

- Use **Disable()** instead of **Delete()** if you want to remove an object temporarily. **Disable()** stops an object from being rendered but allows it to be reused later.
- Nullify references **(ObjectReference = None)** to ensure they don’t linger in memory after they’re no longer needed.
- Garbage collect by making sure your scripts remove all unused references and resources at the end of a function or event

Using External Data and JSON Handling

Papyrus provides ways to handle external data by reading and writing files. This allows you to store complex data, track player progress, configure settings, and create dynamic game mechanics.
One of the best formats for structured data storage is **JSON (JavaScript Object Notation)**, as it allows for easy organization of nested information.

═══★═══════════★═══

1. Why Use External Data?

Using external files in Skyrim modding allows for:
✅ **Persistent storage** – Save player settings or progress outside the save file.
✅ **Mod configuration** – Let users customize features without editing scripts.
✅ **Dynamic events** – Load new game data without modifying the mod.
✅ **Cross-mod communication** – Share data between different scripts.

═══★═══════════★═══

2. Reading and Writing Text Files in Papyrus

Before handling **JSON**, you need to understand how Papyrus interacts with files. Skyrim supports basic text file reading and writing via the `File I/O API`

Example 1: Writing to a File

```
Scriptname WriteToFile extends Quest

Function SaveData()
String filePath = "Data/SKSE/Plugins/MyModData.txt" ; File
location
String data = "Player Gold: " + Game.GetPlayer().GetGoldAmount()

File myFile = File.Open(filePath, File.WRITE)
If myFile
myFile.WriteLine(data)
myFile.Close()
Debug.Notification("Data saved successfully!")
Else
Debug.Notification("Failed to save data.")
EndIf
EndFunction
```

📝 **Explanation:**

- `File.Open()` opens the file in write mode.
- `WriteLine()` stores the player's gold amount.
- The file is saved inside ***`Data/SKSE/Plugins/`***

Example 2: Reading from a File

```
Scriptname ReadFromFile extends Quest

Function LoadData()
String filePath = "Data/SKSE/Plugins/MyModData.txt"
File myFile = File.Open(filePath, File.READ)

If myFile
String savedData = myFile.ReadLine()
myFile.Close()
Debug.Notification("Loaded Data: " + savedData)
Else
Debug.Notification("No saved data found.")
EndIf
EndFunction
```

📝 **Explanation:**

- The script reads data from a file and displays it in a notification.
- `ReadLine()` retrieves one line at a time.

═══★═══════════★═══

3. Handling JSON Data in Papyrus

JSON is useful when you need to store structured data. However, Skyrim's Papyrus does not have native **JSON** support, so we use **JContainers** or **PapyrusUtil**,
two SKSE plugins that extend Papyrus functionality.

Installing JSON Support
To use **JSON handling**, you need:
✅ JContainers SE/AE
✅ PapyrusUtil SE/AE

4. Writing JSON Data (Jcontainers)

JContainers allows you to store complex data in **JSON format**.

Example 3: Saving a Player's Inventory as **JSON**

```
Scriptname SaveInventoryJSON extends Quest

Function SaveInventory()
JMap inventoryData = JValue.Object() ; Create JSON object
Actor player = Game.GetPlayer()
inventoryData.SetStr("PlayerName",
player.GetLeveledActorBase().GetName())
inventoryData.SetInt("Gold", player.GetGoldAmount())

JArray items = JValue.Array()
FormList playerItems = player.GetInventory()

; Loop through inventory
Int i = 0
While i < playerItems.GetSize()
Form item = playerItems.GetAt(i)
JMap itemData = JValue.Object()
itemData.SetStr("ItemName", item.GetName())
itemData.SetInt("ItemCount", player.GetItemCount(item))
items.AddObj(itemData)
i += 1
EndWhile

inventoryData.SetObj("Items", items)

; Save to file
JValue.WriteToFile("SKSE/Plugins/InventorySave.json",
inventoryData)
Debug.Notification("Inventory saved to JSON!")
EndFunction
```

📝 **Explanation:**

- `JMap` stores key-value pairs ***(like a dictionary)***.
- `JArray` stores lists of objects ***(like an array)***.
- `JValue.WriteToFile()` saves the **JSON** data.

5. Reading JSON Data (Jcontainers)

You can reload the saved **JSON** data and use it in your mod.

Example 4: Loading Inventory from **JSON**

```
Scriptname LoadInventoryJSON extends Quest

Function LoadInventory()
JMap inventoryData =
JValue.ReadFromFile("SKSE/Plugins/InventorySave.json")
If inventoryData
String playerName = inventoryData.GetStr("PlayerName")
Int gold = inventoryData.GetInt("Gold")
Debug.Notification(playerName + " has " + gold + " gold.")

JArray items = inventoryData.GetObj("Items")
Int i = 0
While i < items.Length()
JMap itemData = items.GetObjAt(i)
String itemName = itemData.GetStr("ItemName")
Int itemCount = itemData.GetInt("ItemCount")
Debug.Notification("Item: " + itemName + " (x" + itemCount + ")")
i += 1
EndWhile
Else
Debug.Notification("Failed to load inventory data.")
EndIf
EndFunction
```

📝 **Explanation:**

- `JValue.ReadFromFile()` loads the **JSON** file.
- `JMap.GetStr()` and `JMap.GetInt()` retrieve stored values.
- The script loops through saved items and displays them.

6. Using JSON for Mod Configuration

You can use **JSON** to store mod settings that players can customize without editing scripts.

Example 5: Loading Mod Settings from **JSON**

```
Scriptname LoadModConfig extends Quest

Function LoadSettings()
JMap config =
JValue.ReadFromFile("SKSE/Plugins/MyModSettings.json")

If config
Bool enableFeature = config.GetBool("EnableFeature")
Float npcSpeed = config.GetFlt("NPCSpeedMultiplier")
Debug.Notification("Feature Enabled: " + enableFeature)
Debug.Notification("NPC Speed Multiplier: " + npcSpeed)
Else
Debug.Notification("No config file found.")
EndIf
EndFunction
```

📝 **Explanation:**

- The script loads **JSON** mod settings for toggling features or changing values dynamically

Interacting with Havok Physics

Papyrus scripting allows you to control **Havok physics objects** in Skyrim, including projectiles,

doors, movable objects, and dynamic props. By manipulating these elements, you can create realistic interactions, improve immersion, and even add new gameplay mechanics.

═══★═══════════★═══

1. Manipulating Physics Objects in the Game World

Any object in Skyrim that interacts with the **Havok physics engine** can be influenced using Papyrus. These objects include:

- **Moveable Static Objects –** Barrels, carts, crates, etc.
- **Physics-Enabled Items –** Weapons, shields, books, and clutter.
- **Doors and Gates –** Openable doors with physics-based movement.
- **Projectiles –** Arrows, spells, and thrown objects.

To interact with these objects, we use Papyrus functions such as `SetMotionType()`

```
ApplyHavokImpulse()

MoveTo()
```

═══★═══════════★═══

2. Moving and Rotating Objects

You can control movable objects dynamically by adjusting their position or applying physical forces.

Example 1: Making an Object Float in the Air

```
Scriptname FloatingObject extends ObjectReference

Event OnLoad()
Self.SetMotionType(Motion_Keyframed, False) ; Disables physics for
stable floating
Self.MoveTo(Self.GetPositionX(), Self.GetPositionY(),
Self.GetPositionZ() + 100)
EndEvent
```

Explanation:

- `SetMotionType(Motion_Keyframed, False)` disables **Havok physics** so the object remains in place

- The `MoveTo()` function shifts the object **100 units** upward, making it float.

═══★═══════════★═══

3. Applying Force to Objects (Impulses)

To create realistic movement, you can apply force to physics objects using `ApplyHavokImpulse()`

Example 2: Knocking Over a Barrel

```
Scriptname KnockOverBarrel extends ObjectReference

Event OnActivate(ObjectReference akActivator)
Self.ApplyHavokImpulse(0, 0, 50, 10) ; Pushes the object upwards
with force
Debug.Notification("You knock over the barrel!")
EndEvent
```

Explanation:

- When activated, the script applies an impulse ***(force)*** that pushes the barrel upward.
- The numbers **`(0, 0, 50, 10)`** represent **X, Y, Z force values** and **force duration**.
- This can be used for traps, physics-based puzzles, or interactive objects.

4. Controlling Doors and Gates with Physics

Doors and gates can be manipulated using `SetOpen()` and physics impulses.

Example 3: Forcing a Door Open with Magic

```
Scriptname MagicDoorOpener extends ActiveMagicEffect

ObjectReference Property TargetDoor Auto

Event OnEffectStart(Actor akTarget, Actor akCaster)
If TargetDoor
TargetDoor.SetOpen(True)
TargetDoor.ApplyHavokImpulse(0, 0, 100, 5) ; Pushes the door open
Debug.Notification("The door blasts open!")
EndIf
EndEvent
```

Explanation:

- The script forces a door to open when a spell is cast.
- `SetOpen(True)` ensures it unlocks, and `ApplyHavokImpulse()` makes it swing open dynamically.

5. Customizing Projectile Physics

Projectiles ***(arrows, fireballs, thrown objects)*** also use **Havok physics** and can be modified.

Example 4: Altering Arrow Flight with Wind

```
Scriptname WindAffectedArrow extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
Projectile arrow = akCaster.GetEquippedWeapon().AsProjectile()
If arrow
arrow.ApplyHavokImpulse(10, 0, 0, 2) ; Push arrow sideways like
wind
Debug.Notification("The wind affects your shot!")
EndIf
EndEvent
```

Explanation:

- The script alters an arrow's trajectory by applying sideways force.
- This can simulate wind effects or magical interference.

═══★═══════════★═══

6. Making Dynamic Props React to Spells

You can make objects explode, shatter, or move when hit by magic.

Example 5: A Fireball That Blows Objects Away

```
Scriptname FireballExplosion extends ActiveMagicEffect

ObjectReference Property ExplosionObject Auto

Event OnEffectStart(Actor akTarget, Actor akCaster)
ExplosionObject.ApplyHavokImpulse(0, 0, 200, 10) ; Launch object
upwards
Debug.Notification("The explosion sends objects flying!")
EndEvent
```

Explanation:

- The script simulates an explosion effect, launching objects into the air.
- Can be used to add destructible environments or interactive combat mechanics.

7. Making Objects Follow the Player

A physics-enabled object can be set to follow the player dynamically.

Example 6: A Floating Lantern That Follows You

```
Scriptname FloatingLantern extends ObjectReference

Event OnInit()
While True
Self.MoveTo(Game.GetPlayer())
Utility.Wait(1) ; Moves every second to follow the player
EndWhile
EndEvent
```

Explanation:

- The object constantly moves to the player's position, following them.
- Can be used for floating companions, magical artifacts, or lanterns.

═══★═══════════★═══

Custom Spell and Ability Scripting

Papyrus scripting allows you to create unique spells and abilities that introduce new gameplay mechanics and interactions. By modifying magic effects, adding custom scripts, and using conditions, you can create spells that go beyond Skyrim's default magic system.

═══★═══════════★═══

1. Understanding Magic Effects in Papyrus

Each spell or ability in Skyrim is built from **Magic Effects (MGEF)**. These determine how the spell functions, such as dealing damage, healing, summoning, or applying status effects.

Some key Magic Effect Archetypes include:

- **Fire and Frost Damage –** Standard destruction magic.
- **Absorb Health/Magicka/Stamina –** *Drains the target's stats.*
- **Paralysis –** Temporarily disables the target.
- **Summon Creatures –** *Calls an entity to assist the player.*
- **Scripted Effects –** Allows full customization with Papyrus scripts.

By combining different effects and scripts, you can create powerful and unique abilities.

2. Adding Scripted Effects to Spells

Papyrus scripts can be attached to Magic Effects to modify how a spell behaves when cast.

Example 1: A Fireball That Pushes Enemies Back

```
Scriptname FireballPushEffect extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
If akTarget
akTarget.PushActorAway(akCaster, 10) ; Push enemy back
Debug.Notification("The target is knocked back!")
EndIf
EndEvent
```

Explanation:

- When the spell hits an enemy, it applies a knockback effect using `PushActorAway()`
- This creates a unique destruction spell that mimics a fire explosion with force.

═══★═══════════★═══

3. Creating Custom Abilities

Abilities differ from spells in that they are passive or triggered by specific conditions.

Example 2: A Passive Health Regeneration Ability

```
Scriptname PassiveRegenEffect extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
While akTarget.HasMagicEffect(Self)
akTarget.RestoreActorValue("Health", 5)
Utility.Wait(2) ; Heals 5 HP every 2 seconds
EndWhile
EndEvent
```

Explanation:

- When the effect is applied, the target regenerates 5 HP every 2 seconds.
- This can be used for a racial ability or a custom enchanted item.

4. Applying Custom Conditions to Spells

You can use conditions to make spells trigger only under certain circumstances.

Example 3: A Lightning Spell That Does Extra Damage in Rain

```
Scriptname WeatherBoostedLightning extends ActiveMagicEffect

Spell Property LightningSpell Auto

Event OnEffectStart(Actor akTarget, Actor akCaster)
If Weather.GetCurrentWeather().GetClassification() == 2 ; 2 =
Rainy Weather
akTarget.DamageActorValue("Health", 50) ; Extra damage
Debug.Notification("The storm empowers your lightning!")
Else
akTarget.DamageActorValue("Health", 25) ; Normal damage
EndIf
EndEvent
```

Explanation:

- The spell checks if the current weather is rainy before applying extra damage.
- This adds environmental interaction to magic.

═══★═══════════★═══

5. Spells That Scale with Player Attributes

A spell can scale based on the caster’s skill level, making it more powerful over time.

Example 4: A Fireball That Grows Stronger with Destruction Skill

```
Scriptname ScalingFireball extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
Float destructionLevel = akCaster.GetActorValue("Destruction")
Float damageAmount = 10 + (destructionLevel * 0.5) ; Base 10 + 0.5
per skill point
akTarget.DamageActorValue("Health", damageAmount)
Debug.Notification("Your fireball deals " + damageAmount + "
damage!")
EndEvent
```

Explanation:

- The spell calculates its damage based on the Destruction skill level of the caster.
- This makes magic more rewarding as the player levels up.

═══★═══════════★═══

6. Summoning and Custom AI Behavior

You can create unique summoning spells with customized AI behavior.

Example 5: Summon a Ghost That Follows and Protects You

```
Scriptname SummonGuardianEffect extends ActiveMagicEffect

Actor Property SummonedGuardian Auto

Event OnEffectStart(Actor akTarget, Actor akCaster)
Actor ghost = akCaster.PlaceAtMe(SummonedGuardian)
ghost.SetPlayerTeammate(True)
Debug.Notification("A ghostly warrior appears to aid you!")
EndEvent
```

Explanation:

- The spell spawns a ghost NPC that fights alongside the player.
- `SetPlayerTeammate(True)`-makes it follow and assist in combat.

═══★═══════════★═══

7. Removing Custom Effects

To ensure spells do not last indefinitely, you can remove them after a set time.

Example 6: A Temporary Speed Boost

```
Scriptname SpeedBoostEffect extends ActiveMagicEffect

Event OnEffectStart(Actor akTarget, Actor akCaster)
akCaster.ModActorValue("SpeedMult", 50) ; Increases speed
Utility.Wait(10) ; Lasts 10 seconds
akCaster.ModActorValue("SpeedMult", -50) ; Resets speed
EndEvent
```

Explanation:

- Increases movement speed for 10 seconds, then resets it

═══★═══════════★═══

AI Package Manipulation with Scripts

AI (Artificial Intelligence) packages in Skyrim determine how NPCs behave under different conditions. By using Papyrus scripts, you can dynamically modify these **AI packages** to control NPC reactions, movement, and interactions based on in-game events. This allows for more immersive and responsive NPC behavior.

1. Understanding AI Packages

AI packages define an NPC's actions, such as:

- **Wander** – The NPC moves randomly within a defined area.
- **Follow** – The NPC follows a specific target ***(e.g., the player)***.
- **Eat/Sleep/Work** – The NPC engages in daily activities.
- **Combat** – The NPC engages in fights under specific conditions.
- **Flee** – The NPC runs away when threatened.

By manipulating these packages, you can dynamically change an NPC's behavior using scripts.

═══★═══════════★═══

2. Applying AI Package Changes in Scripts

Papyrus provides functions to assign and modify AI packages at runtime. The most commonly used functions are:

- `MyNPC.SetPackageOverride(MyPackage)` → Forces an NPC to use a specific package.
- `MyNPC.EvaluatePackage()` → Forces the NPC to immediately re-evaluate their AI behavior.
- `MyNPC.SetPlayerTeammate(True/False)` → Makes the NPC follow the player.

Example 1: Making an NPC Follow the Player on Command

```
Scriptname NPCFollowScript extends ObjectReference

Actor Property MyNPC Auto
Package Property FollowPlayerPackage Auto

Event OnActivate(ObjectReference akActivator)
If akActivator == Game.GetPlayer()
MyNPC.SetPackageOverride(FollowPlayerPackage)
MyNPC.EvaluatePackage()
Debug.Notification("The NPC is now following you!")
EndIf
EndEvent
```

Explanation:

- When the player activates an object ***(e.g., a lever)***, the NPC is assigned a Follow package.
- `EvaluatePackage()` ensures the NPC updates their behavior immediately.

═══★═══════════★═══

3. Temporarily Overriding AI Packages

You can override an NPC's default behavior for a short time and later restore their original AI package.

Example 2: Making an NPC Flee When Attacked

```
Scriptname NPCFleeScript extends Actor

Package Property FleePackage Auto
Package Property DefaultPackage Auto

Event OnHit(ObjectReference akAggressor, Form akSource, Projectile
akProjectile, bool abPowerAttack, bool abBashAttack, bool
abHitBlocked)
SetPackageOverride(FleePackage)
EvaluatePackage()
Utility.Wait(10) ; NPC will flee for 10 seconds
SetPackageOverride(DefaultPackage)
EvaluatePackage()
EndEvent
```

Explanation:

- When the NPC is hit, they switch to a **Flee AI package**.
- After 10 seconds, they return to their original behavior.

═══★═══════════★═══

4. Using AI Manipulation for Quests

AI package manipulation is useful in quests to ensure NPCs act according to the story.

Example 3: Making an NPC Walk to a Specific Location

```
Quest Property MyQuest Auto
Actor Property TargetNPC Auto
Package Property WalkToMarkerPackage Auto

Function StartEscortMission()
If MyQuest.GetStage() == 10 ; Check if the quest is at the right
stage
TargetNPC.SetPackageOverride(WalkToMarkerPackage)
TargetNPC.EvaluatePackage()
Debug.Notification("The NPC is now walking to their destination.")
EndIf
EndFunction
```

Explanation:

- When the quest reaches stage 10, the NPC starts walking to a predefined location.

═══★═══════════★═══

5. Removing AI Package Overrides

To reset an NPC's behavior to their default AI package, use:

```
MyNPC.ClearPackageOverride()
MyNPC.EvaluatePackage()
```

This ensures the NPC resumes their standard routine.

═══★═══════════★═══

Conclusion

Papyrus opens endless possibilities—keep experimenting, keep creating. Your journey is just beginning.

Common Questions (FAQ) about Papyrus Scripting

Q: What is Papyrus, and why should I learn it?

A: *Papyrus is the scripting language used in Skyrim's Creation Kit. If you want to create quests, modify game mechanics, or enhance mods with custom functionality, learning Papyrus is essential.*

Q: Do I need programming experience to learn Papyrus?

A: *Not necessarily! While programming knowledge helps, Papyrus is relatively simple compared to other languages. With practice and a structured guide (like this one!), you can learn it step by step.*

Q: How do I create a new script in the Creation Kit?

A: *Open the Creation Kit, go to the "Gameplay" menu, select "Papyrus Script Manager", and click "New Script". Name your script, choose a parent script if needed, and start coding.*

Q: Why isn't my script compiling?

A: *There could be multiple reasons:*

- ***Syntax errors*** *(check for missing brackets or semicolons).*
- ***Missing script dependencies*** *(ensure required scripts are present).*
- ***Papyrus compiler not set up correctly*** *(verify your SkyrimEditor.ini settings).*

Q: Where can I find Papyrus script documentation?

A: *The best sources are:*

- ***The Creation Kit Wiki*** *(if still available).*
- *Online modding communities like* ***Nexus Mods forums*** *or* ***r/skyrimmods*** *on Reddit.*

Q: Can I edit existing scripts from Skyrim?

A: *Yes, but with limitations. Many base game scripts are* ***compiled (.PEX files)****. To edit them, you need the* ***uncompiled (.PSC) source files****, which can be extracted using tools like* ***Champollion****.*

Credits

Written by: ***ElysTabby***

This guide wouldn't have been possible without the incredible ***Skyrim modding community****. A special thanks to all the modders, scripters, and contributors who have shared their knowledge, insights, and tools over the years. Your dedication continues to inspire and push the boundaries of what's possible in Skyrim.*

Whether through tutorials, forum discussions, or groundbreaking mods, your contributions have made learning Papyrus scripting a much more accessible and enjoyable journey. This guide is a small way to give back to the community that has given so much.

Thank you for your passion, creativity, and endless support!❤