Multiplayer Animal Controller for Fish-Net – Documentation

Version 1.1

Disclaimer

This asset is a **script-only extension** to **MalberS Animal Controller** designed to add multiplayer functionality using **Fish-Net Networking**. It allows syncing of gameplay over the network for objects based on the MalberS Animal Controller, facilitating the development of multiplayer games.

- **Dependencies:** This asset requires the **MalberS Animal Controller** and **Fish-Net Networking**.
- **Usage and Modification:** You are free to use and modify this asset as required for your projects. The scripts are designed to integrate with the MalberS Animal Controller, but modifications may be necessary depending on your use case.
- **No Guarantees:** Every effort has been made to ensure the asset works as expected, but **no guarantees** are provided regarding performance in all situations or projects. It is strongly recommended to fully test the asset within your specific project before using it in production.

Overview

This asset extends the **MalberS Animal Controller** to integrate multiplayer functionality using **Fish-Net Networking**. It allows you to sync gameplay over the network, ensuring that players' actions, animations, and object states are properly communicated between the server and clients in a multiplayer environment.

To use this asset effectively, it is important to have a base understanding of **Multiplayer Networking** in Unity. Some familiarity with **Fish-Net** (or similar networking frameworks like Unity's Netcode) will greatly help in implementing and troubleshooting the networked features this asset provides.

Key concepts to familiarize yourself with include:

- **Server-Client Architecture:** Some understanding how the server and clients communicate is crucial. The server typically manages game logic and authoritative actions, while clients handle player-specific interactions.
- **Network Ownership:** Properly assigning and managing network ownership is key to ensuring that actions, such as picking up items or triggering player-specific events, are only performed by the correct player.

We recommend reviewing the Fish-Net documentation to get an understanding of how to set up and use networking in Unity. You can check out the full Fish-Net documentation at: Fish-Net Documentation

What is needed to be synced and handled to make Multiplayer work is already in place within this asset along with Demo Scene to connect two local clients to test and play with. What is currently not covered with in the Demos of this asset is Menus and Scenes require to make the Multiplier fully ready to distribute as this is something that will need to be setup for your specific needs and game.

Installation

1. **Import the Package:**
   - Make sure MalberS Animal Controller is Imported
     1. AC 1.4.8+
     2. HAP 4.4.8+
   - Make sure the Network Scripts are imported
     1. Fish-Net V4+
     2. PurrNet V1.8+
     3. Unity NGO is being considered as a alternative
   - Download and import the package into your Unity project using the **Unity Asset Manager**.
   - Once imported, you will find the Prefabs, Demos and Scripts within the Integration Folder in MalberS Animal Controller
2. **Prerequisites:**
   - Ensure that **MalberS Animal Controller** and **Fish-Net Networking** are already set up in your project.
   - If you are new to **Fish-Net Networking**, please refer to the Fish-Net documentation for proper setup and configuration.

Usage Guide

1. Script and Component needed based on Object

- Animal Controller object, Require the following
    - Networking Scripts
        - Network Object
        - Network Transform
        - Network Animator
    - MalbarS Scripts Replaced
        - Aim, replaced with NetAim
        - Damageable, replaced with NetDamageable
        - Rider, replaced with NetRider
        - Weapon Manager, replaced with Net Weapon Manager
        - Pick Up – Drop, replaced with Net Pick up – Drop
    - Scripts to add from this Asset
        - Animal Instance
            - Added to the same location as the Animal
            - There is also a Player Instance that should be used for Player Objects
        - Stat Sync
            - Added to the same location as Stats and Damageable
        - Collectable Storage
            - Added to the same location as Animal Instance
- Moveable Objects
    - Networking Scripts
        - Network Object
        - Network Transform
    - MalbarS Scripts Replaced
        - If Pickable, replace with NetPickable
    - Scripts to add from this Asset
        - Ownership Sync
- Weapon Objects
    - The Bellow information is for "Holster Weapons" with out Inventory
    - Networking Scripts
        - Network Object
        - Network Transform
    - Scripts to add from this Asset
        - Ownership Sync

        - Weapon Utility
            - This comes in a few types pick the one that fits for the Weapon.
- Interactable Objects
    - This can vary from object to object but here are some of the Utilities that will help
    - Network Scripts
        - Network Object
    - Scripts to add from this Asset
        - Component Sync
        - Trigger Event Sync
        - Zone Utility
        -

2. Script Usage

- **Client Instance**
    - This script is in essence the player, it also makes the up the main Player Object. Both the Script and it Game Object make a great location to store player data during the session as the Animal Object can be delete and owner ship changed.
- **Component Sync**
    - Used to make sure Components are on or off depending if they are on the server or client. Example Platforms: we turn on the moving script only on the servers so that way it controls the position to all clients.
- **Trigger Event Sync**
    - Is more of Utility Script that other look for, this is placed at the room of an object with the Network Object providing some other Non-Network Behavior Utilities access to network calls
- **Zone Utility**
    - This Addon Utility handles some of the extra tasks for syncing MalberS “Zone” scripts across the network. It also serves as the check for if the Object is a player and handles all the UI type call and event rather than the Zone Script.
- **Ownership Sync**
    - This Network Behavior handles the ownership of Objects it is attached to that you may want to change owner to give clients control over
    - This is used for anything picked up or controlled by the players as Ownership need to be passed over to the client for better movement control.
- **Stat Sync**

    - This is to enable the syncing of stats between the Server and Clients, Health, Stamina, Air? All of it as it changes will get synced over.
- **Animal Instance**
    - This is the main script to manage the whole “Animal” Object, it enables and disable what is need and sync what is required for owner to operate the “Animal”
        - Player Instance is used for the Player Characters, each Client should only have control of 1 of these at a time.
- **Weapon Utility Scripts**
    - There are 3 scripts for this group each make sure the needed actions are synced for combat based on the weapon type they are used with.
    - **Unarmed**
        - This Script sync the activation of the colliders required for damage with an Unarmed type of attack.
    - **Melee / Ranged**
        - These Pair are responsible for weapon based attacks and adds “Player Events” for UI calls
- **Replace / Inherited Scripts**
    - **NetAim**
        - Replaces Aim this script used to sync the aim of “Animal”
    - **NetWeapon Manager**
        - Replaces Weapon Manager, handles the
    - **Net Damageable**
        - Replaces Damageable, make sure damage is only handled on Server but allows reaction to damage to happen on the Owners System to make sure that the animal reacts correctly to damage
    - **Net Pickable**
        - Replaces Pickable and aids in the handling of owner ship and pickup sync between clients and server
    - **Net Pickup Drop**
        - Replaces Pickup Drop, the other half of the pickup scripts required to make interaction work correctly and synchronize
    - **Net Rider**
        - Replaces Rider and manages the Rider Script for network actions
    - **Net Mount**
        - Replaces Mount used to handle ownership hand-off of mounts so that the rider can own the mount and give that client full control.

One key Feature of all these replacement Script is that they have “Player Events”, these events are meant to fire only when the call to them relate to a Player Object and that Call is happening on the Player Objects Owner’s computer. This is where all UI Event calls should be done to avoid them happening on other clients when you do not want them.

Using the needed scripts is the same as their counter parts in MalberS, please review their Documentation

https://malbersanimations.gitbook.io/animal-controller

Registered Scriptable Objects

Registered Scriptable Objects (RSO) is a utility to aid in the syncing of Non-Networked Objects, main use is for Scriptable Objects but also supports Game Objects

RSOManager is a Singleton and the call location for retrieval of both the RSO and Keys for data transfer.

RSO Database is a Scriptable Object itself and a storage location for all the elements to be index for easier access. This database store Scriptable Objects and can search for the key stored in relation to it.

RSO Root, Link and Game Object are used to speed up the look up of Objects as it add key Value with a simple GetKey() to retrieve it to transfer the ID to the client/server when needed.

RSOGameObject, store their values with a Key Value to speed up search but not required for Scriptable Objects.

Scriptable Objects can be added to the Database by right clicking on the asset and selecting “Add to RSO Library”.

To access the data from scripts call the following

RSOManager.Get(int key, bool returnLink = false)

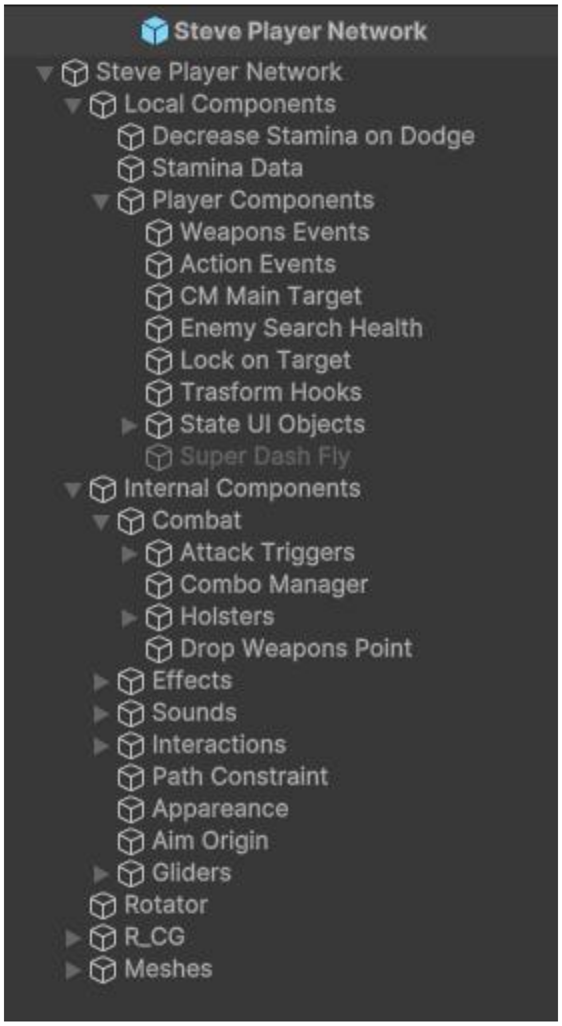
This Function returns the Scriptable Object linked to the int key.
returnLink will return the Scriptable Object RSOLink if it is setup as such, when false it returns the Value stored in RSOLink.

RSOManager.GetKey(ScriptableObject)

This Function returns the int Key for the related scriptable object, if it is a RSORoot object it will pull the key from the from there rather than searching the Database for the Key.

Example Screen Shots

1. Animal Game Object Hierarchy



```
Steve Player Network
▼ Steve Player Network
    ▼ Local Components
        Decrease Stamina on Dodge
        Stamina Data
        ▼ Player Components
            Weapons Events
            Action Events
            CM Main Target
            Enemy Search Health
            Lock on Target
            Trasform Hooks
            ▶ State UI Objects
            Super Dash Fly
    ▼ Internal Components
        ▼ Combat
            ▶ Attack Triggers
            Combo Manager
            ▶ Holsters
            Drop Weapons Point
        ▶ Effects
        ▶ Sounds
        ▶ Interactions
        Path Constraint
        Appareance
        Aim Origin
        ▶ Gliders
    Rotator
    ▶ R_CG
    ▶ Meshes
```

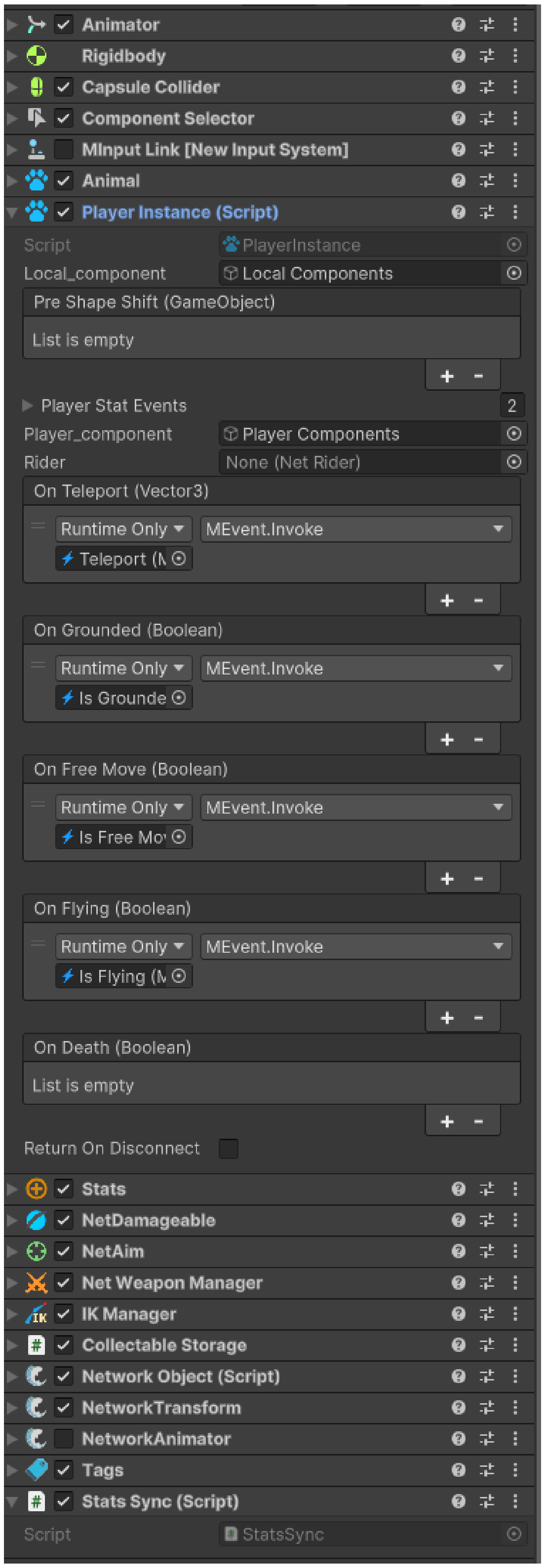
This Example taken from the Player Character show an optimal Hierarchy for Animal Objects.

The Root is setup the same MalberS but there are 3 Key Sub Objects

1. Internal Components
These are all the components need for the Game Object to work on both Server and Clients.
2. Local Components
These items are needed for the Owner of the Object to function this includes some logic items like "Stamina on Dodge"
3. Player Components
These are all the UI and Control, Objects and Events needed for the owner of the Animal to control its actions, it is setup as its own Object under Local Components in the event that a player object has to sit idle while someone reconnects and the Server is a Host running both client and server.

Player and Local Components are enabled and Disabled by Animal Instance.

2. Animal Game Object Root Components



Animator

Rigidbody

Capsule Collider

Component Selector

MInput Link [New Input System]

Animal

Player Instance (Script)

Script: PlayerInstance

Local_component: Local Components

Pre Shape Shift (GameObject)

List is empty

Player Stat Events 2

Player_component: Player Components

Rider: None (Net Rider)

On Teleport (Vector3)

Runtime Only | MEvent.Invoke | Teleport (N

On Grounded (Boolean)

Runtime Only | MEvent.Invoke | Is Grounde

On Free Move (Boolean)

Runtime Only | MEvent.Invoke | Is Free Mo

On Flying (Boolean)

Runtime Only | MEvent.Invoke | Is Flying (N

On Death (Boolean)

List is empty

Return On Disconnect ☐

Stats

NetDamageable

NetAim

Net Weapon Manager

IK Manager

Collectable Storage

Network Object (Script)

NetworkTransform

NetworkAnimator

Tags

Stats Sync (Script)

Script: StatsSync

This is an Example pull from the Player Animal Object, with the Player Instance open.

This Shows Player and Local Components linked to the script along with some of the Animal Events that require some Global Invokes, such calls meant to affect the Player and UI need to be call only on the Client that has ownership.

Please Check out the Steve Player Network Prefab.

Additional Features

MalberS Horse Aimset Pro (HAP) Support

This asset supports **MalberS Horse Aimset Pro** (HAP). However, to prevent compile issues, **preprocessor directives** are used to disable HAP support by default. Users who wish to enable HAP support will need to add the **Scripting Define Symbol** “AC_HAP” to their project settings.

Luxus-1’s PushPull State Support

This asset also supports **Luxus-1’s PushPull State**. To enable this support, users need to add the **Scripting Define Symbol** “AC_PUSHPULL” to their project settings.

Example Prefabs for HAP and PushPull State

Included in the package are example prefabs for both **MalberS Horse Aimset Pro (HAP)** and **Luxus-1’s PushPull State**, demonstrating how these features can be integrated and synchronized in a networked environment.

Unity Netcode for Game Objects Support (Experimental)

This asset also has the base required scripts to support NGO in the future, there are currently some key differences between Fish-Net and NGO and limitation to NGO that are making some features of MalberS Animal Controller difficult to sync using NGO.

Once these issues are resolved NGO support can be activated similarly to the above additions.

Scripting Define Symbol – Location

- In Unity Menus Select “Edit” -> “Project Settings”
- In Player Tab look for “**Scripting Define Symbol**”
- Press the “+” and type in the needed text.
    - Make sure to press apply when done adding