Plugins

Nali Weapons 3 Final

1 – Introduction

This mod comes with the possibility of adding plugins which change or enhance some of the pack behavior, either at client or server level, or both. To do so you need some knowledge of Unreal Script, and how to create new packages.

To start, you need to create a subclass from the class *NWMutator*, and setup the *bLoadMutatorForClient* property in the *defaultproperties* to *True* or *False* depending if you want your mutator to run on the client or not.
In case you set the *bLoadMutatorForClient* to True, do not forget to add you final package to the ServerPackages list in case you intend to use it in a server, and if not, it's extremely advisable to NOT add it to that list at all, specially if it's a security plugin.

Next just pick the function(s) you want to use to modify any behavior at all from the pack. These functions are listed below in ***Functions***.

After you made your code, be sure to compile your new package with the *NWCoreVIII.u* package, otherwise it won't compile since the plugin needs to create a dependency to this pack (since the *NWMutator* class only exists there).
In the end, after you have created your package, just load it as a regular mutator in order to work (create a normal mutator *.int* file if you intend to release it to public for standalone enjoyment by players).

2 – Functions

Here are listed all the available functions you can use from the *NWMutator* class in order to make your plugin.
Always be sure to keep the chain of plugins intact by keeping the original function code or calling a *"return Super.someFuntion(params);"* at the end of it.

Function: *isValidZeroPingPreCheck*	**Return:** *bool*	**Arguments:** 11 *[+2 optional]*
Network: Server	**Denomination:** Zero Ping security plugin	

Description:

This function is called before the Zero Ping main checks during a shot, and returns True when the shot is valid.
If the shot is not valid, the function should return *False* and *useStandardShot* should be set to *1* in case it's intended for the shot to be processed as a normal ping affected trace.

Arguments:
isAltFire – If *True*, the shot was made using the secondary fire mode of the weapon;
NW – Weapon the shot was made from;
Other – Actor that was hit by the trace in the client;
HitLoc – Trace hit location;
Start – Trace start location;
OtherClientLoc – Other location reported by the client;
OwnerClientLoc – Weapon owner location reported from by client;
OwnerClientVRot – Weapon owner view rotation reported by the client;
Accuracy – Weapon current shot accuracy;
AccY – Weapon Y axis trace accuracy;
AccZ – Weapon Z axis trace accuracy;
useStandardShot – Output parameter to indicate if the weapon should still attempt to process a normal trace from the server (*0 – False*, *1 – True*);
ignoreFlags – Output parameter to indicate the trace validations the weapon should ignore as binary flags (for a complete list of these flags, see the ***Zero Ping*** section of *NW3_SDK.pdf*).

```
function bool isValidZeroPingPreCheck(private bool isAltFire, private NaliWeapons NW, private
Actor Other, private vector HitLoc, private vector Start, private vector OtherClientLoc, private
vector OwnerClientLoc, private rotator OwnerClientVRot, private float Accuracy, private float
AccY, private float AccZ, optional private out byte useStandardShot, optional out int
ignoreFlags)
{
	if (NextNWMutator != None)
		return NextNWMutator.isValidZeroPingPreCheck(isAltFire, NW, Other, HitLoc,
			Start, OtherClientLoc, OwnerClientLoc, OwnerClientVRot, Accuracy,
			AccY, AccZ, useStandardShot, ignoreFlags);
	return True;
}
```

Function: *isValidZeroPingPostCheck*	**Return:** *bool*	**Arguments:** 11 *[+1 optional]*
Network: Server	**Denomination:** Zero Ping security plugin	

Description:

This function is called after the Zero Ping main checks during a shot, and returns True when the shot is valid.
If the shot is not valid, the function should return *False* and *useStandardShot* should be set to *1* in case it's intended for the shot to be processed as a normal ping affected trace.

Arguments:
isAltFire – If *True*, the shot was made using the secondary fire mode of the weapon;
NW – Weapon the shot was made from;
Other – Actor that was hit by the trace in the client;
HitLoc – Trace hit location;
Start – Trace start location;
OtherClientLoc – Other location reported by the client;
OwnerClientLoc – Weapon owner location reported from by client;
OwnerClientVRot – Weapon owner view rotation reported by the client;
Accuracy – Weapon current shot accuracy;
AccY – Weapon Y axis trace accuracy;
AccZ – Weapon Z axis trace accuracy;
useStandardShot – Output parameter to indicate if the weapon should still attempt to process a normal trace from the server (*0 – False*, *1 – True*).

```
function bool isValidZeroPingPostCheck(private bool isAltFire, private NaliWeapons NW, private
Actor Other, private vector HitLoc, private vector Start, private vector OtherClientLoc, private
vector OwnerClientLoc, private rotator OwnerClientVRot, private float Accuracy, private float
AccY, private float AccZ, optional private out byte useStandardShot, optional out int
ignoreFlags)
{
	if (NextNWMutator != None)
		return NextNWMutator.isValidZeroPingPreCheck(isAltFire, NW, Other, HitLoc,
			Start, OtherClientLoc, OwnerClientLoc, OwnerClientVRot, Accuracy,
			AccY, AccZ, useStandardShot, ignoreFlags);
	return True;
}
```

Function: *getZHackBehavior*	**Return:** *int*	**Arguments:** 1
Network: Client	**Denomination:** Renderer ZHack behavior	

Description:

This function is called whenever the PlayerPawn renders a weapon or other in its Canvas, and its the decision maker if either or not should the renderer have its *ZHack* property on or off.

It should return one of the following:

-1 – Automatic (let the NW3 system decide based on its *ZHack* setting);
0 – Set forcefully *ZHack* to *False*;
1 – Set forcefully *ZHack* to *True*.

Arguments:

PP – The *PlayerPawn* this side of the network belongs to.

```
simulated function int getZHackBehavior(PlayerPawn PP)
{
	if (Role == ROLE_Authority && NextNWMutator != None)
		return NextNWMutator.getZHackBehavior(PP);
	else if (Role < ROLE_Authority && NextNWClientMutator != None)
		return NextNWClientMutator.getZHackBehavior(PP);
	return -1;
}
```

Function: *isMonsterGame*	**Return:** *int*	**Arguments:** 0
Network: Server	**Denomination:** Monster game check	

Description:

This function is called to know if the current game is a monster game (like Monster Hunt for example), and it should return one of the following:

-1 – Automatic (let the NW3 system decide);
0 – Indicate that this is not a monster game;
1 – Indicate that this is indeed a monster game.

Arguments:

No arguments.

```
function int isMonsterGame()
{
	if (NextNWMutator != None)
		return NextNWMutator.isMonsterGame();
	return -1;
}
```

Function: *isFriend*	**Return:** *int*	**Arguments:** 1 *[+6 optional]*
Network: Server	**Denomination:** Friend check	

Description:

This function is called to know if the pawn *P* is considered to be a “friend” (in another words, a pawn that should not be hurt or attacked).

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that pawn *P* is not a friend;
1 – Indicate that that pawn *P* is indeed a friend.

Arguments:
P – The pawn that should be checked to indicate if it's a friend or not;
Instig – An instigator or caller (like a pawn who's attacking or looking to relative pawn *P*);
team – A team number to of the instigator;
bNoHurtTeam – A setting from the instigator part that may indicate that it's not supposed to hurt teammates;
bNoHurtInstig – A setting from the instigator part that may indicate that it's not supposed to hurt itself;
ownerName – Instigator player name;
src – The actor this function is being called from.

```
function int isFriend(Pawn P, optional Pawn Instig, optional byte team, optional bool
bNoHurtTeam, optional bool bNoHurtInstig, optional string ownerName, optional Actor src)
{
	if (NextNWMutator != None)
		return NextNWMutator.isFriend(P, Instig, team, bNoHurtTeam, bNoHurtInstig,
			ownerName, src);
	return -1;
}
```

Function: *isSameHorde*	**Return:** *int*	**Arguments:** 2
Network: Server	**Denomination:** Monster horde check	

Description:

This function is called to know if the pawn *P* is considered to be in the same “horde” as pawn *Instig*.
By “horde” what is meant is generally 2 monsters belonging to the same “group” of sort to speak.
It's generally called in monster games, and it should return:

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that pawn *P* is not in the same horde as *Instig*;
1 – Indicate that that pawn *P* is indeed in the same horde as *Instig*.

Arguments:
P – The main pawn to check;
Instig – The secondary pawn to check as the instigator of this call.

```
function int isSameHorde(Pawn P, Pawn Instig)
{
	if (NextNWMutator != None)
		return NextNWMutator.isSameHorde(P, Instig);
	return -1;
}
```

Function: *isTeamMember*	**Return:** *int*	**Arguments:** 2
Network: Server or/and client	**Denomination:** Same team check	

Description:

This function is called to know if the pawn *P* is considered to be in the same team as the pawn *Instig*.
It's generally called in team games, and it should return:

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that pawn *P* is not in the same team as *Instig*;
1 – Indicate that that pawn *P* is indeed in the same team as *Instig*.

Arguments:
P – The main pawn to check;
Instig – The secondary pawn to check as the instigator of this call.

```
simulated function int isTeamMember(Pawn PSource, Actor A)
{
	if (Role == ROLE_Authority && NextNWMutator != None)
		return NextNWMutator.isTeamMember(PSource, A);
	else if (Role < ROLE_Authority && NextNWClientMutator != None)
		return NextNWClientMutator.isTeamMember(PSource, A);
	return -1;
}
```

Function: *getTeam*	**Return:** *int*	**Arguments:** 1
Network: Server or/and client	**Denomination:** Get team number from actor	

Description:

This function should return the team number of actor *A*, and it should return:

-1 – Automatic (let the NW3 system decide);
0 to 255 – Team number.

Arguments:
A – The actor to check.

```
simulated function int getTeam(Actor A)
{
	if (Role == ROLE_Authority && NextNWMutator != None)
		return NextNWMutator.getTeam(A);
	else if (Role < ROLE_Authority && NextNWClientMutator != None)
		return NextNWClientMutator.getTeam(A);
	return -1;
}
```

Function: *isValidTarget*	**Return:** *int*	**Arguments:** 1 *[+1 optional]*
Network: Server or/and client	**Denomination:** Valid target check	

Description:

This function is called to know if the actor *A* is valid as an attacking target (for lock, as enemy, etc), and it should return:

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that actor *A* is not a valid target;
1 – Indicate that that actor *A* is indeed a valid target.

Arguments:
A – The actor to check;
ignoreStationaryPawn – Flag to indicate that stationary pawns are intended to be ignored.

```
simulated function int isValidTarget(Actor A, optional bool ignoreStationaryPawn)
{
	if (Role == ROLE_Authority && NextNWMutator != None)
		return NextNWMutator.isValidTarget(A, ignoreStationaryPawn);
	else if (Role < ROLE_Authority && NextNWClientMutator != None)
		return NextNWClientMutator.isValidTarget(A, ignoreStationaryPawn);
	return -1;
}
```

Function: *isMonster*	**Return:** *int*	**Arguments:** 1
Network: Server or/and client	**Denomination:** Monster check	

Description:

This function is called to know if the pawn *P* is a monster or not.

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that pawn *P* is not a monster;
1 – Indicate that that pawn *P* is indeed a monster.

Arguments:
P – The pawn to check.

```
simulated function int isMonster(Pawn P)
{
	if (Role == ROLE_Authority && NextNWMutator != None)
		return NextNWMutator.isMonster(P);
	else if (Role < ROLE_Authority && NextNWClientMutator != None)
		return NextNWClientMutator.isMonster(P);
	return -1;
}
```

Function: *checkFilters*	**Return:** *int*	**Arguments:** 2 *[+1 optional]*
Network: Server	**Denomination:** Replacement filtering check	

Description:

This function is called from a NW3 replacement class (*NWReplacer* subclasses) whenever a weapon, ammo or pickup is about to get replaced, and that item has its *Filters* set to *C=SomeID*, and decides if either or not the item should be replaced, by returning:

-1 – Automatic (let the NW3 system decide);
0 – Indicate that that this replacement cannot be made;
1 – Indicate that that this replacement can be made.

Arguments:
ID – The ID of the replacement;
Filters – The current Filters for this replacement;
Other – The pawn the item may be given to after the replacement.

```
function int checkFilters(string ID, string Filters, optional Pawn Other)
{
	if (NextNWMutator != None)
		return NextNWMutator.checkFilters(ID, Filters, Other);
	return -1;
}
```

Function: *SetPropertiesOptions*	**Return:** *(void)*	**Arguments:** 2
Network: Server	**Denomination:** Properties setup on item	

Description:

This function is called from a NW3 replacement class (*NWReplacer* subclasses) whenever a weapon, ammo or pickup is replaced, and that item has its *Options* set to *C=SomeID*.

Arguments:
ID – The ID of the property setup;
A – The actor that should have its properties set.

```
function SetPropertiesOptions(string ID, Actor A)
{
	if (NextNWMutator != None)
		NextNWMutator.SetPropertiesOptions(ID, A);
}
```

Function: *ignoreSkillKill*	**Return:** *bool*	**Arguments:** 2
Network: Server	**Denomination:** Skill kills validation and modification	

Description:

This function is called from a NW3 skills class (*NWSkillKillsManager*) whenever a skill is processed and validated, and that skill has *C=SomeID*.
It should return *True* in case the skill is valid, and *False* if otherwise.

Arguments:
ID – The ID of the skill;
Msg – The current message of the skill (can be directly modified);
denyList – The current deny list of indexes of the skill (can be directly modified);
extraPoints – The current extra points of the skill (can be directly modified);
extraFrags– The current extra frags of the skill (can be directly modified).

```
function bool ignoreSkillKill(string ID, out string Msg, out string denyList, out int
extraPoints, out int extraFrags)
{
	if (NextNWMutator != None)
		return NextNWMutator.ignoreSkillKill(ID, Msg, denyList, extraPoints,
		extraFrags);
	return False;
}
```

Function: *LoadedDynamicProfile*	**Return:** *(void)*	**Arguments:** 1
Network: Server	**Denomination:** Profile dynamic load event	

Description:

This function is called when a new server/gameplay profile is loaded.

Arguments:
ProfileID – The ID of the profile loaded.

```
function LoadedDynamicProfile(byte ProfileID)
{
	if (NextNWMutator != None)
		NextNWMutator.LoadedDynamicProfile(ProfileID);
}
```