Warno Modding Manual

Version 2.2.2

Table of Contents

Creating a new mod

Go to the game installation directory (in steam: Library → WARNO → Properties → Local Files → Browse Local Files), and enter the `Mods` subdirectory.

Open a console in the *Mods* directory (either by using `cd <path_to_Mods_directory>` or by shift + right click in windows explorer > "open console here") and run *CreateNewMod.bat* with the mod name as argument.

Example:

```
CreateNewMod.bat MyAwesomeMod
```

The name must be unique ; it will not work if a directory already exists with that name. If all goes well, you should see a message indicating that the mod was created, and a new directory named *MyAwesomeMod* has appeared. It should contain the following directories:

- CommonData
- GameData

and the following files:

- base.zip
- CreateAssetDefinitionFiles.bat
- CreateModBackup.bat
- GenerateMod.bat
- RetrieveModBackup.bat
- UpdateMod.bat
- UploadMod.bat

These 7 files (*base.zip* and the 6 *.bat*) are part of the modding tools, please do not modify them or you will likely compromise your mod.

If something is missing, try to delete the mod directory and recreate it.

You are now able to start working on your mod.

Generating a mod

A generation step is needed before a mod can be activated in-game or uploaded to the Steam Workshop. To proceed, launch *GenerateMod.bat* in your mod directory.

Once you have generated your mod a default SteamID will be given to it. You need to upload your mod to give him a real SteamID. Even if you haven’t finish your work, you can upload it and hide it on your workshop mod page.

Editing a mod configuration file

Go to the *C:\Users\USERNAME\Saved Games\EugenSystems\WARNO\mod\MyAwesomeMod* directory, and open the Config.ini file:

```
Name = MyAwesomeMod ; The name of your mod
Description = My Mod is Awesome ; The description of your mod
TagList = ; Don't edit that field
PreviewImagePath = C:\my_mod_image.png ; An image located on your hard drive
CosmeticOnly = 0 ; Does your mod affect gameplay or not ? If activated, the mod will
not be shared when creating a lobby
ID = 902495510 ; Mod Steam ID, do not modify Version = 0 ; Value to increment when an
update in this mod is incompatible with the current version
DeckFormatVersion = 0 ; Increment this value to invalidate all decks for this mod
ModGenVersion = 0 ; Don't edit that field
```

If you want to change the mod name, you have to : change the "Name" field in the configuration file, change the directory name in *C:\Users\USERNAME\Saved Games\EugenSystems\WARNO\mod*, change the directory name in the steam directory.

The "ID" and "TagList" are filled when UploadMod.bat is launched.

Uploading a mod

To upload your mod on steam, launch *UploadMod.bat* in your mod directory. The screen will become black for a short period of time.

This process will update `TagList`, `ID` and `ModGenVersion` fields.

Once you’ve uploaded your mod, it’s ready to use on steam workshop, you can access its own page, set its visibility, description, image, etc…

Updating a mod – When a new game patch is released

With the game updating through Steam, it will be needed from time to time to update your mod so that it still work with new versions. We created a small tool to help you with this, requiring minimal manual work. Before talking about it, let’s see what the problem is.

Why do we need to update mods?

Imagine you have a mod that modifies file *A.ndf*. When this mod was created, the original file (from game data) was:

```
Thing is TThing
```

```
(
    SomeInt = 42
)
```

And you modified it to be:

```
Thing is TThing
(
    SomeInt = 43
)
```

No problem here, that's the point of modding. Now what if we at Eugen need to modify this file as well, and end up adding content to it so it looks like this in the new version of the game:

```
Thing is TThing
(
    SomeInt = 42
)
Thing2 is TThing
(
    SomeInt = 80
)
```

You will want to keep your change to `Thing/SomeInt` value, and have the new `Thing2` as well. Wanted result for your mod:

```
Thing is TThing
(
    SomeInt = 43
)
Thing2 is TThing
(
    SomeInt = 80
)
```

Updating mods

The solution is quite easy, just run *UpdateMod.bat* in your mod directory. This will merge your changes with the last version of the game data.

If you're lucky, the process will work automatically and display a message indicating success. However, it is possible that you encounter conflicts during the operation. Conflicts arise when both the new version of the game and your mod modify the same part of a NDF file. In this case, the process can't guess what result your want, and need manual intervention.

Solving conflicts

Imagine once again that you're modifying *A.ndf*.

Original version (game data when the mod was created):

```
Thing is TThing
(
    SomeInt = 42
)
```

Modded version:

```
Thing is TThing
(
    SomeInt = 43
)
```

And with the following update, the game version of the file becomes this:

```
Thing is TThing
(
    SomeInt = 80
)
```

Both the game and the mod have modified the same line. You can easily see that it is entirely up to you what final value should `Thing/SomeInt` have in your mod, and that no tool can decide it on its own.

The updating tool will stop and warn you about the files that contain conflicts. To handle this case, the tool will mark conflicts like this one with special delimiters:

```
Thing is TThing
(
<<<<<<<
    SomeInt = 80
|||||||
    SomeInt = 42
=======
    SomeInt = 43
>>>>>>>
)
```

- After `<<<<<<<` is the new version from the game data
- After `|||||||` is the base common ancestor (in our case original game data)
- After `=======` is your mod version

- `>>>>>>>` marks the end of the conflict

Resolving the conflicts consists in modifying the file so that it makes sense to the game that will load it (ie. get rid of the delimiters and have a single `SomeInt = …` line) and has the wanted value for your mod. You are free to keep your mod value (43), to go back to the original value (42), to get the new value (80) or to decide an entirely new.

The process is very similar to a three-way merge like found in version control software such as git.

Mods backup

Inside your mod directory you will find 2 essentials scripts:

- CreateModBackup.bat : which will allow you to create a backup of the current status of your mod. Backups are basically a copy of your mod files archived and stored in *Backup/* directory. A backup will be generated every time you upload your mod, considering that if you upload your mod that means that you're currently on a stable version. You can generate a backup yourself by using this script it will ask for a name, you can leave the entry empty it will use a generated name based on current time.
- RetrieveModBackup.bat : this script will allow you to reset your mod to a previous version stored in *Backup/*, it will simply copy and force paste the backup version in your current working space so be careful before any RetrieveModBackup. The script will prompt you for a name, you can leave the entry empty it will select the last backup generated.

NDF Map

Every files location you could want to edit.

Interface

If you want to mod the interface, you gonna need to look at GameData/UserInterface/.

All files under *Style/* directory will be related to global objects and reused style across different screens.

All files under *Use/* will describe each screen or part of screen, separated in directory following their usage in game :

- InGame : everything after a match session is launch
- OutGame : everything related to main menu, loading, etc…
- Common : everything common to both In and OutGame.
- ShowRoom : everything related to Armory and Deck Creation menu
- Strategic : everything related to Army General mode

Gameplay

Units

All unit descriptors are located in *GameData/Generated/Gameplay/Gfx/UniteDescriptor.ndf*. You will find various other interesting descriptors in the same directory, like *WeaponDescriptor.ndf* and *Ammunition.ndf*.

Divisions

Divisions are located in *GameData/Generated/Gameplay/Decks/Divisions.ndf*. Divisions are composed by packs, a pack is a bundle of the same units, all packs are located in *GameData/Generated/Gameplay/Decks/Packs.ndf*.

Gameplay constants

GameData/Gameplay/Constantes/ contains all files defining generic gameplay features, for example:

- in *GDConstantes.ndf* you will find the property `DefaultGhostColors` which define the color for ghosts
- `RelativeBonusMoneyByIADifficulty` allow you to give more income money to the AI.

Weapons

Weapons are mainly described in *WeaponDescriptor.ndf* and *Ammunition.ndf*, located in *GameData/Generated/Gameplay/Gfx*.

The first file describe, for each units, how its turrets are configured, and which weapons are mounted. For example, you'll find the properties defining the number of salvos for each weapon, the angle ranges and rotation speeds of the turrets, and some UI-related parameters. You will also find reference to `TAmmunitionDescriptor` objects, which are defined in *Ammunition.ndf*.

In these objects there are many useful properties, here are some of them:

- `Arme`: the type of damage the weapon will inflict. For AP weapon, you can use values from `AP_1` to `AP_30` (see *CommonData/Gameplay/Constantes/Enumerations/ArmeType.ndf*).
- `PorteeMaximale`: range of the weapon.
- `HitRollRuleDescriptor`: describes how the hit and pierce chance are computed (see *GameData/Gameplay/Constantes/HitRollConstants.ndf*).
- `IsHarmlessForAllies`: enables/disables friendly fire.

AI

GameData/Gameplay/Skirmish contains files about the AI. An artificial intelligence is called strategy.

There are multiple strategies (used in skirmish, multiplayer, operations and army general) depending on game difficulty and settings, located in *GameData/Gameplay/Skirmish/Strategies/* subdirectories.

Strategies are plugged in the *GameData/Gameplay/Skirmish/IASkirmishDescriptors.ndf* file.

A strategy contains a list of Generators and Transitions. The AI navigates from one

`TSequenceGeneratorDescriptor` to another depending on `TIAGeneralStrategyTransition`.

A `TSequenceGeneratorDescriptor` describes the AI's priorities while it is active. MacroAction behavioral descriptors used in the Generators are located in *GameData/Gameplay/Skirmish/SkirmishMacros* directory. Most MacroActions control unit groups with specific objectives (attack, defend, etc.). The required units for each action are described in TPoolModel objects which are freely modifiable. Note: new TPoolModel objects need distinct GUID fields.

Global settings about the AI are defined in *GameData/Gameplay/Constantes/IAStratConstantes.ndf* and affect both skirmish and campaign AI.

Creating your first mod, step-by-step

This section objective is to create a mod which allows to create decks with more units (100 slots points instead of 50).

Follow instructions of the first section of this manual, called **Creating a new mod**, to create your mod `MyAwesomeMod`.

From your mod directory, open file *GameData/Generated/Gameplay/Decks/Divisions.ndf*.

Replace all `MaxActivationPoints = 50` to `MaxActivationPoints = 100`.

Follow instructions of section **Generating a mod** to generate your mod.

You can now activate and test your mod, in the game through the menu `Mod Center`.

Follow instructions of section **Uploading a mod**.

Congratulation, you have created and published your first mod in WARNO workshop.

Asset integration

When creating a new mod, some necessary NDF files are created for your mod to support asset integration.

However, if you want to add assets to a mod created before asset integration through modding was possible (v.130173, date from 27/08/2024), you need to run once *CreateAssetDefinitionFiles.bat* in your mod directory.

Localisation

Text displayed in game for unit names, interface buttons etc. is generated from CSV files.

You should find 5 .csv files in your subdirectory *GameData/Localisation/%YOUR_MOD_NAME%/*. They all have same structure, but with different purpose :

- *COMPANIES.csv* is used for Army General companies names.

- *INTERFACE_INGAME.csv* is used for interface during tactical battle and army general game.
- *INTERFACE_OUTGAME.csv* is used for interface of every out-game menu (main menu, armory, deck creation menu, etc.)
- *PLATOONS.csv* is used for Army General platoons names.
- *UNITS.csv* is used for tactical unit names.

For each file, every line after the first one creates a localisation token, and contains 2 columns:

- `TOKEN` indicates its name referenced in NDF. It cannot contains more than 10 characters, and must be unique.
- `REFTEXT` is associated text.

It is recommended to use a text editor (Notepad, Sublime Text, etc.) to edit those files.

If you want to use a CSV editor like LibreOffice, **make sure you save your file using semi-colons `;` as delimiters.** It is the only character our generation reads as a delimiter.

Specific case: unit first names

Unit first names are generated from a localisation file depending of their nationality. Those files are defined in *UnitNames.ndf*:

```
unnamed TLocalisationDicoResource
(
    DicoToken = "NAMES_US"
    FileName = "GameData:/Gameplay/Unit/Names/UNITNAMES_US.csv"
)
```

You can create your own if you want to define names for a new nationality. They have the same structure than other CSV localisation file, with a `TOKEN` and a `REFTEXT` column, but each `TOKEN` must be a number starting from `1`.

For instance, here are the 10 first lines of our own *UNITNANMES_US.csv*:

```
"TOKEN";"REFTEXT"
"1";"Abbott"
"2";"Acosta"
"3";"Adkins"
"4";"Aguirre"
"5";"Ali"
"6";"Alpine"
"7";"Alvarado"
"8";"Alvarez"
"9";"Anthony"
```

Translations for supported languages

`REFTEXT` is the text used for "English" language, the game default language.

By default, it will also be used for any other supported language accessible to players.

You can add specific translations for other languages by adding lines in *GameData/Localisation/Database/Translations.csv*. (Create this file if it doesn't exist already). It should have 4 columns, in this order:

- `TOKEN` must be filled with `TOKEN` column in your base localisation file.
- `LANG` indicates your translation target. It must be one of those :
  - `FR` for French.
  - `GER` for German.
  - `POL` for Polish.
  - `RU` for Russian.
  - `SC` for Simplified Chinese.
  - `SPA` for Spanish.
- `REFTEXT` must be filled with `REFTEXT` column in your base localisation file.
- `TEXT` is the text used for your token in that language.

Example

Let's create a custom unit name.

In *UNITS.csv*, add a new row with two columns : `TJPHQZYPUG` for the token name, `A cool new unit` for text.

In text mode, your line should look like this:

```
TJPHQZYPUG;A cool new unit
```

Or like this, depending if you enforced quoted fields:

```
"TJPHQZYPUG";"A cool new unit"
```

Both formatting are valid.

Now we need to plug our new token in NDF to use it. In *GameData/Generated/Gameplay/Gfx/UniteDescriptor.ndf*, pick a unit to rename. You can change its name by changing value of `NameToken` parameter of its `TUnitUIModuleDescriptor`. Replace it by your new token: `'TJPHQZYPUG'`.

Generate your mod with *GenerateMod.bat*, launch the game with your mod activated, go to the armory, and search for a unit named "A cool new unit". You should find the unit you renamed.

Optionaly, we can add specific translations for it. For that, we need to add a new line in *GameData/Localisation/Database/Translations.csv*:

```
"TJPHQZYPUG";"FR";"A cool new unit";"Une nouvelle unité cool"
```

Regenerate your mod with *GenerateMod.bat*. Now, your unit should be named "Une nouvelle unité cool" if your game language is French, and "A cool new unit" for any other language.
You can add more translations for your unit name by adding more line in *Translations.csv* with different `LANG`.

Texture

You can integrate textures by putting your own .png files in your mod GameData subdirectory and referencing them in NDF.

However, **try to put your .png files into a subdirectory hierarchy similar to vanilla game files !** for instance, our flag textures path is *GameData:/Assets/2D/Interface/Common/Flags/*, so if you want to create your own flag texture, make a *GameData/Assets/2D/Interface/Common/Flags/* subdirectory to your mod, and put your .png file inside it.

Example

Let's replace US flag texture by a new one.

Flags textures are referenced in *GameData/UserInterface/Use/Common/CommonTextures.ndf*.

Look for `CommonTexture_MotherCountryFlag_US` which is US flag texture declaration.

Replace file path `GameData:/Assets/2D/Interface/Common/Flags/US_FLAG.png` by another file name with similar path, like `GameData:/Assets/2D/Interface/Common/Flags/MY_OWN_FLAG.png`.

Now, in your mod directory, create subdirectory hierarchy *GameData/Assets/2D/Interface/Common/Flags/*, and save a custom RGB file named *MY_OWN_FLAG.png* inside.

Generate your mod with *GenerateMod.bat*, launch the game with your mod activated, and go to the armory. You should now see your image instead of US flag, both for "filter by country" buttons and US unit cards.

Mesh

You can integrate new static meshes from .fbx files exported from Blender, version 4.0 or later.

Our meshes have a maximum bone count of 96 and a maximum vertex count of 65536.

FBX export

Export must be done with very specific options. On File → Export → FBX :

- On Include menu
  - `Custom Properties` checkbox must be checked.
- On Transform menu
  - `Apply Scalings` option must be set on `FBX All`.
  - `Forward` option must be set on `Y Forward`.

    - `Up` option must be set on `Z Up`.
- On Geometry menu
    - `Triangulate Faces` checkbox must be checked.
- On Armature menu
    - `Add Leaf Bones` checkbox must be unchecked.

Textures & Material configuration

A single mesh can have several materials. But those materials have strict rules on Shading workspace:

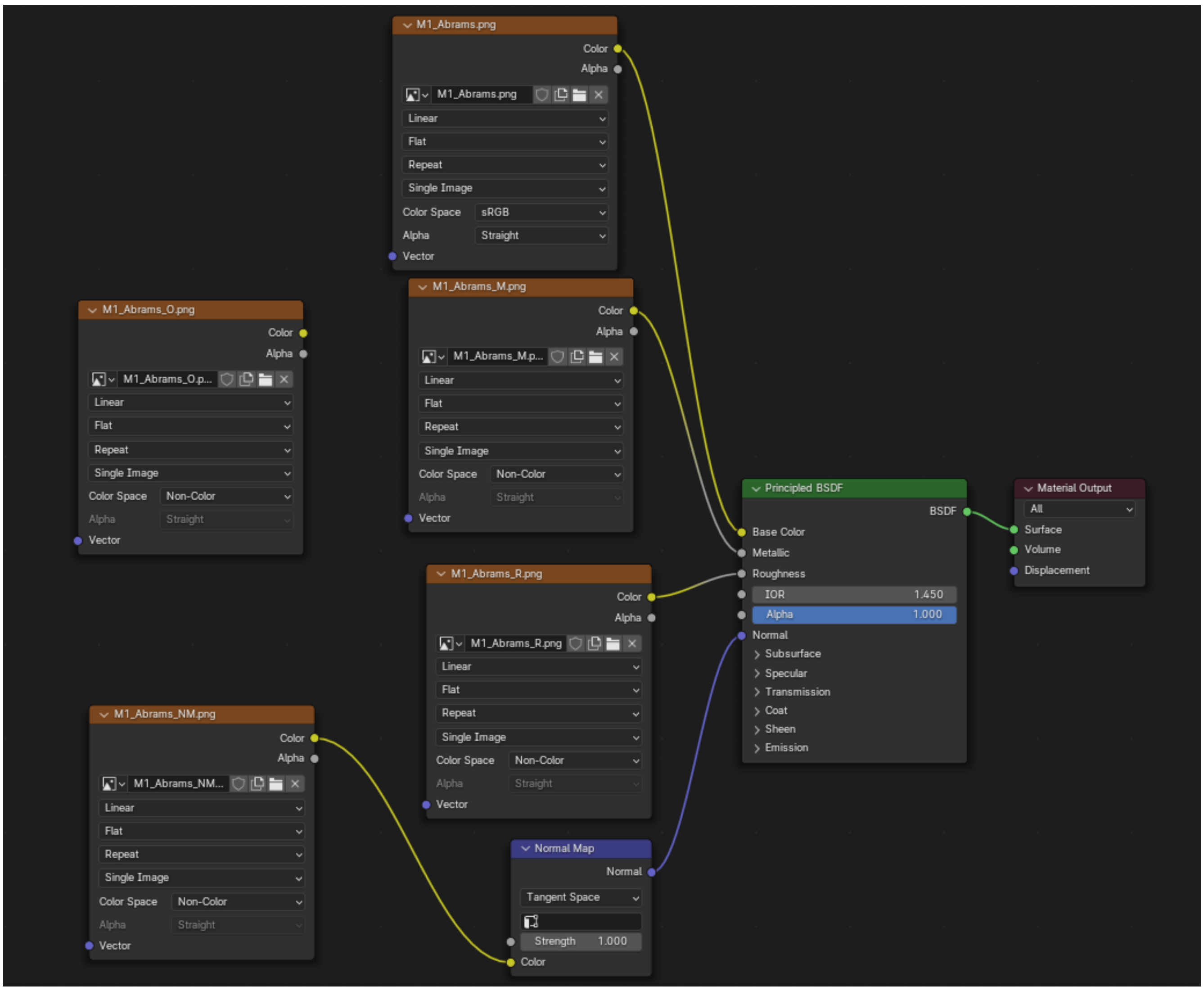
A material must have a RGB `Image Texture` node directly plugged in `Base Color` material surface.

Other optional textures can be plugged :

- A RGB NormalMap `Image Texture` can be plugged through a `Normal Map` node in `Normal` material surface. Its file name must end with `_NM.png`.
- A grayscale Metalness `Image Texture` can be plugged in `Metallic` material surface. Its file name must end with `_M.png`.
- A grayscale roughness `Image Texture` can be plugged in `Roughness` material surface. Its file name must end with `_R.png`.
- A grayscale occlusion `Image Texture` can exists, but to be exported must be set as a material Custom Properties named `EugAO`. Its file name must end with `_O.png`.
- For specific cases like vehicle tracks or transparent materials, a grayscale alpha `Image Texture` can be plugged in `Alpha` material surface. Its file name must end with `_A.png`.

NOTE Only use an alpha texture if you **want** transparency in your material. If your mesh part is opaque, it is better to not use an alpha texture at all.

As an example, this is our M1_Abrams main material configuration :


M1_Abrams.png
Color
Alpha
M1_Abrams.png
Linear
Flat
Repeat
Single Image
Color Space sRGB
Alpha Straight
Vector

M1_Abrams_M.png
Color
Alpha
M1_Abrams_M.p...
Linear
Flat
Repeat
Single Image
Color Space Non-Color
Alpha Straight
Vector

M1_Abrams_O.png
Color
Alpha
M1_Abrams_O.p...
Linear
Flat
Repeat
Single Image
Color Space Non-Color
Alpha Straight
Vector

M1_Abrams_R.png
Color
Alpha
M1_Abrams_R.png
Linear
Flat
Repeat
Single Image
Color Space Non-Color
Alpha Straight
Vector

M1_Abrams_NM.png
Color
Alpha
M1_Abrams_NM...
Linear
Flat
Repeat
Single Image
Color Space Non-Color
Alpha Straight
Vector

Normal Map
Normal
Tangent Space
Strength 1.000
Color

Principled BSDF
BSDF
Base Color
Metallic
Roughness
IOR 1.450
Alpha 1.000
Normal
Subsurface
Specular
Transmission
Coat
Sheen
Emission

Material Output
All
Surface
Volume
Displacement

Every texture must be a .png file.

Every texture must be located in the exported .fbx file directory.

Specific case : continuous track material

Some vehicles need a specific material for their tracks, in order to bind a specific shader able to rotate UV indices when the vehicle is moving.

In order to use this specific track shader, name of material must contains either "*Chenille_Gauche*" (for left track) or "*Chenille_Droite*" (for right track).

Correct unit skeleton

To be considered a valid unit mesh, an exported mesh must have its main part (possibly its only one) called "Chassis". Every other part name is optional.

A unit mesh parts react to specific gameplay events : when a tank aims on its right side, its main turret turns clockwise. When it fires, its canon recoils, a fire FX starts and its body slightly recoils, etc.

This is possible because meshes are separated in a precise part hierarchy, whose names are referenced by "operators" in NDF, mostly in *GameData/Gameplay/Gfx/Units/DepictionOperators.ndf*.

In order to use your mesh for a unit, you must either :

- Ensure it has every bone asked for in NDF. Those bones don't need vertices, they can be empty, bust they must be present.
- Disable `Operators` and `Actions` referencing bone names in your unit depiction definition. This is made by emptying `Operators` and `Actions` for entry referenced by your unit depiction.

You can use meshes located in *Mods/Meshes/* as examples of valid unit bone hierarchy : they are meshes used in WARNO.

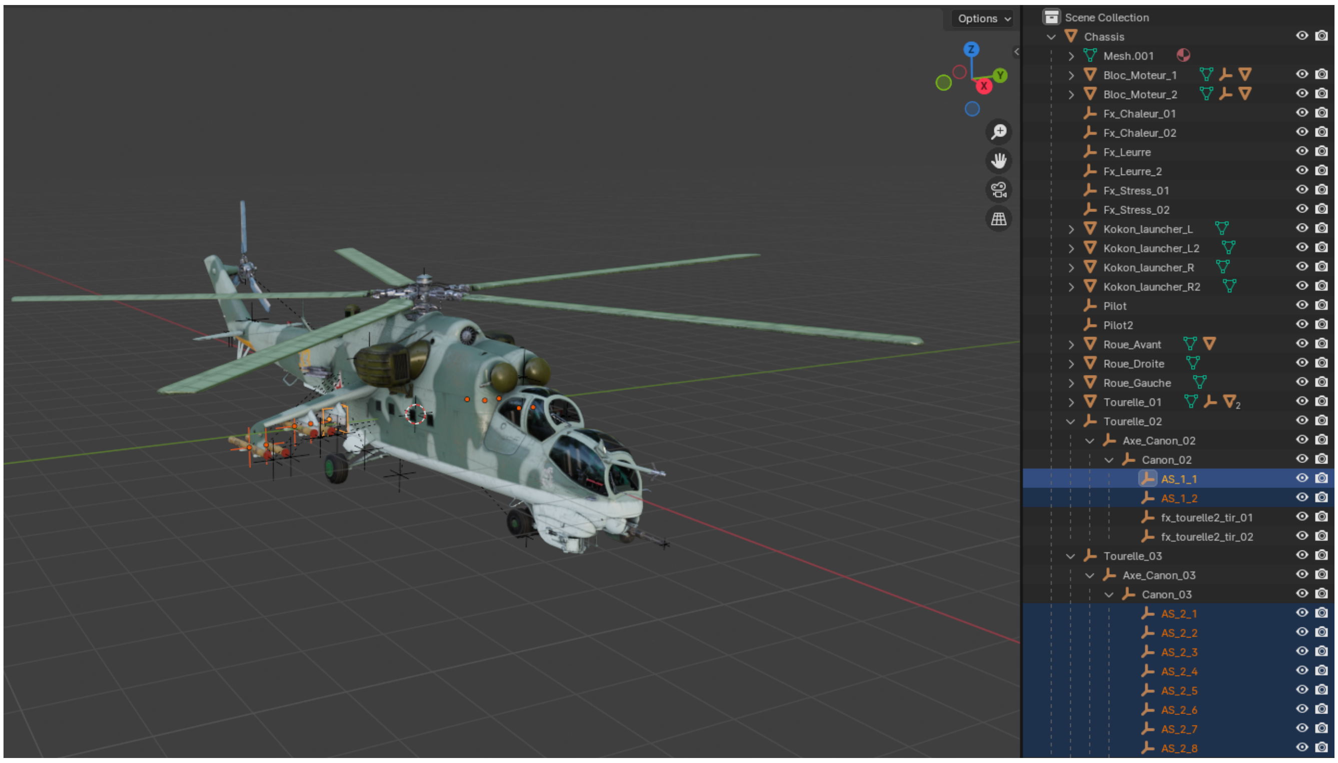
Specific case : missiles attachment locations

Some helicopters and airplanes units have weapons needing hardpoints to attach missiles or rocket pods. Such hardpoints must be specified on model with specific bones.
Those bones have a specific naming convention:

- `AS_{N}_{M}` for air-to-air OR air-to-ground weapons.
- `AA_{N}_{M}` for air-to-air weapons.

`{N}` and `{M}` must be replaced by numbers : `{N}` being weapon number, and `{M}` being missile number.

As an example, our MI_24V_ATGM model can host two weapons needing hardpoints : one with up to 2 missiles or rocket pod, and another with up to 8 missiles or rocket pod :



Example

Let's replace M1_Abrams_US mesh by a new one.

Referenced meshes for this unit are specified in *GameData/Gameplay/Gfx/DepictionResources/US/Char/M1_Abrams_US.ndf*. There are three meshes :

- `Modele_M1_Abrams_US`, the main one, used in armory and at very close range.
- `Modele_M1_Abrams_US_MID`, less detailed, used at camera mid-range.
- `Modele_M1_Abrams_US_LOW`, simplistic, used when the unit is very far away from camera.

Replace main model file path `GameData:/Assets/3D/Units/US/Char/M1_Abrams/M1_Abrams.fbx` by another file name with similar path, like `GameData:/Assets/3D/Units/US/Char/MyNewMesh/MyNewMesh.fbx`.

Now, in your mod directory, create subdirectory hierarchy *GameData/Assets/3D/Units/US/Char/MyNewMesh/*, move your .blend and associated .png files inside it, then export your .fbx file following **FBX export** and **Textures & Material Configuration** instructions. Remember your mesh main part must be named "Chassis", and your materials need at least an associated base color image file.

Then, unless your mesh has every bone needed for such a tank, let’s disable references to specific bone names.

First, we need to locate depiction of this unit. In *GameData/Generated/Gameplay/Gfx/UniteDescriptor.ndf*, search for `Unit_M1_Abrams_US` entity descriptor. In associated `ApparenceModelModuleDescriptor`, we find its `Depiction` parameter : `Gfx_M1_Abrams_US_Autogen`.

`Gfx_M1_Abrams_US_Autogen` definition is in *GameData/Generated/Gameplay/Gfx/Depictions/GeneratedDepictionVehicles.ndf*.

Without bones, we have to empty `Operators` and `Actions` lists by replacing them with empty lists :

```
Operators = []
Actions = MAP []
```

It means your mesh won’t move or start any FX on movement, fire, aiming etc. But it cannot do it without specific bones.

Now, everything should be ready. Generate your mod with *GenerateMod.bat*, launch the game with your mod activated, go to the armory, and look for the *M1 ABRAMS*. Unit card picture is still the same, but when selecting it, you should see your mesh instead of vanilla Abrams mesh.

NOTE Since you didn’t replace MID and LOW meshes, the game will still use vanilla meshes during battle unless the camera is very close from your unit. You can use your own mesh at any range by replacing `Modele_M1_Abrams_US_MID` and `Modele_M1_Abrams_US_LOW` file paths, but beware : replacing too many mid and low meshes by highly detailed meshes shall decrease game performance.

Sound

Sounds and musics can be added through .wav or .ogg files.
For either format, recommanded sampling rate is 48 000 Hz.
Musics and unit voices (refered in engine as "acknows") must be exported as **Stereo**, while spatialised sounds (engine sounds, canon sounds, etc) must be exported as **Mono**.

Musics can be added by modifying either `Playlist_Outgame` or `Playlist_Ingame` in *GameData:/Sound/PerpetualMusic.ndf*.
Acknows can be modified in *GameData:/Generated/Sound/AcknownDescriptors.ndf*.
Interface sounds and SFX are plugged in various files.

Example

Let’s replace the sound played when entering the armory from the main menu.
Most of menu sounds are listed in *GameData:/UserInterface/Use/Common/UICommonResources.ndf*.
Look for `HoverSound_hub_armory` declaration.
Replace file path `GameData:/Assets/Sounds/OutGame/hub_armory.wav` by another file name with similar path, like `GameData:/Assets/Sounds/OutGame/MyOwnSound.wav`.

Now, in your mod directory, create subdirectory hierarchy *GameData/Assets/Sounds/Outgame/*, and export a custom *MyOwnSound.wav* inside.

Generate your mod with GenerateMod.bat, launch the game with your mod activated, and go to the armory. You should hear your sound on entering.

Video

Startup introduction video and loading videos can be replaced by your own.
Videos must be WEBM format, with one video track encoded VP9, and one optional sound track using Vorbis codec.
.webm files must be stored somewhere in your mod GameData directory.

Loading videos can be replcaed in *GameData:/UserInterface/Use/OutGame/UIOutGameLoadingResources.ndf*.
Introduction video can be replaced in *CommonData:/LogoVideos/Intro.ndf*.
If you plug you own subtitles for introduction video, subtitles are localisation tokens to define in *INTERFACE_OUTGAME.csv*.

Example

Let’s replace videos in loading screen by a new one.
In *UIOutGameLoadingResources.ndf*, replace the entire list of `PossibleVideos` by a list of one `TResourceVideo` referencing your own .webm file.

Generate your mod with GenerateMod.bat, launch the game with your mod activated. You see your own video rather than vanilla loading videos on each loading screen.