Modders Guide to ENDLESS Legend II

August, 2026
ver 1.0

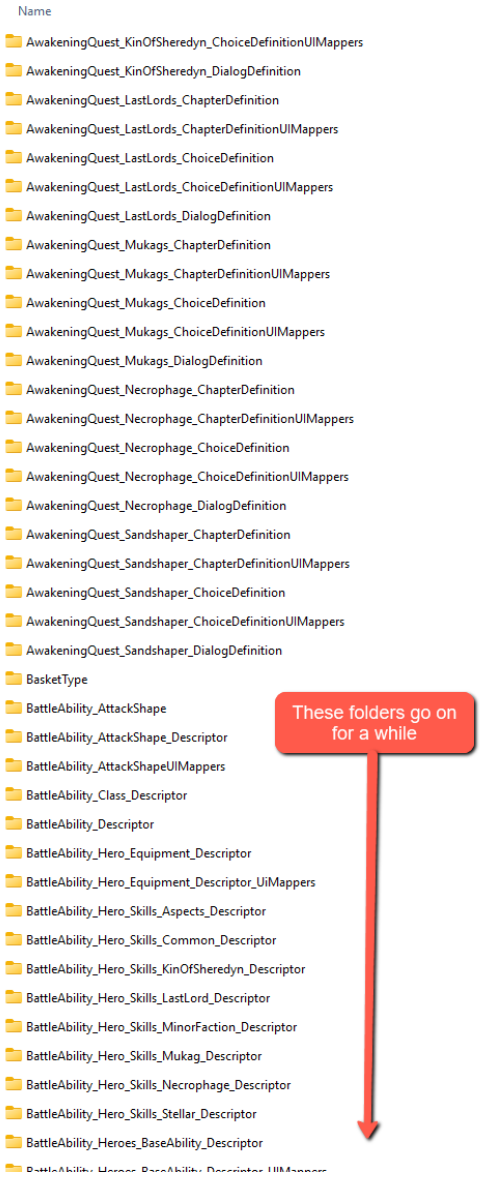
How Modding Works

All the games data is exported as json files. It's a LOT of data. Somewhere around 30,000 files. They are separated into folders by type. At first glance it's an overwhelming cacophony of entries, links and strange values.

We have opted to give you everything. Even though a lot of the information is incredibly difficult to understand we prefer that you have it rather than limit it to just data we think is easier to mod. We are leaning on the side of powerful use over ease of use.

By copying these files you can modify an existing asset (a unit, faction trait, quest reward, pretty much anything) or create a new asset entirely.

The export is included with every build of ENDLESS Legend 2 in the game's folder "..\ENDLESS Legend 2\Public\Modding\assets_data.zip".

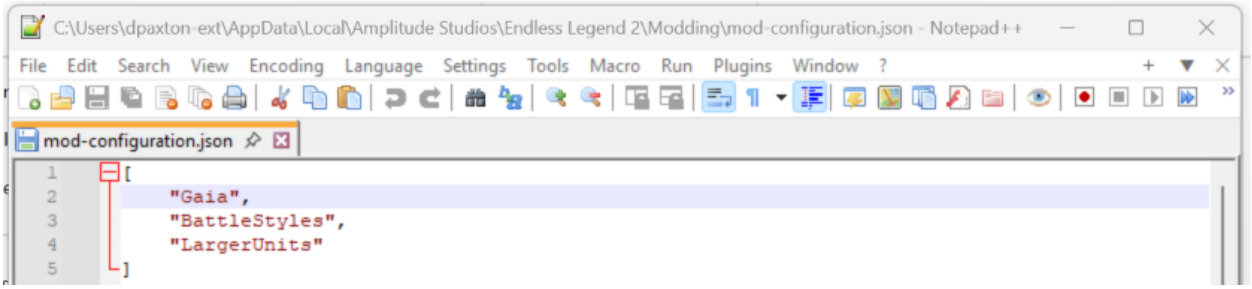


Creating a Mod

Making a mod is simple.

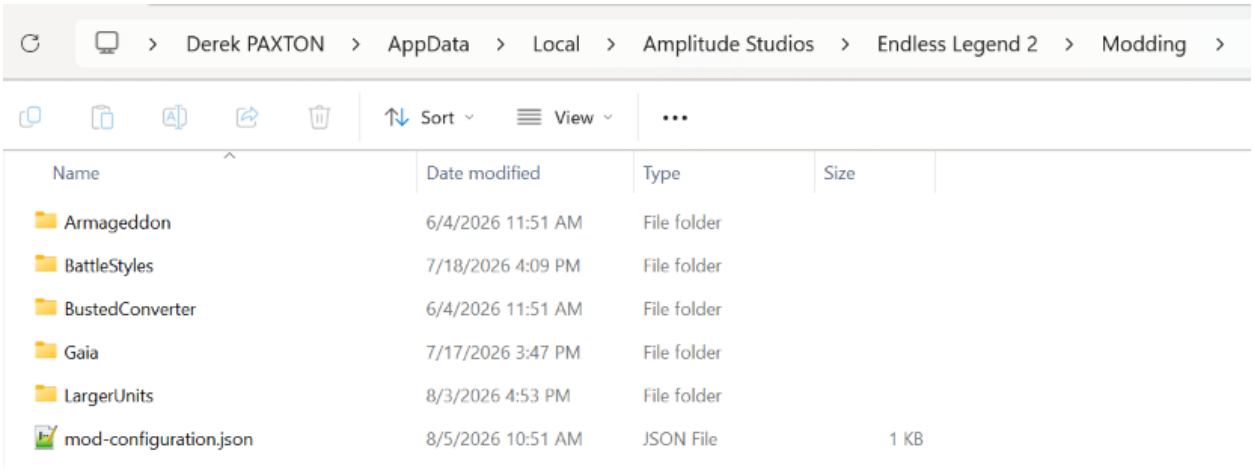
1. In your %localappdata%\Amplitude Studios\Endless Legend 2 directory, create a “Modding” directory.

2. In your Modding directory you need to have a mod-configuration.json file with the name of your active local mods. It should look like this:

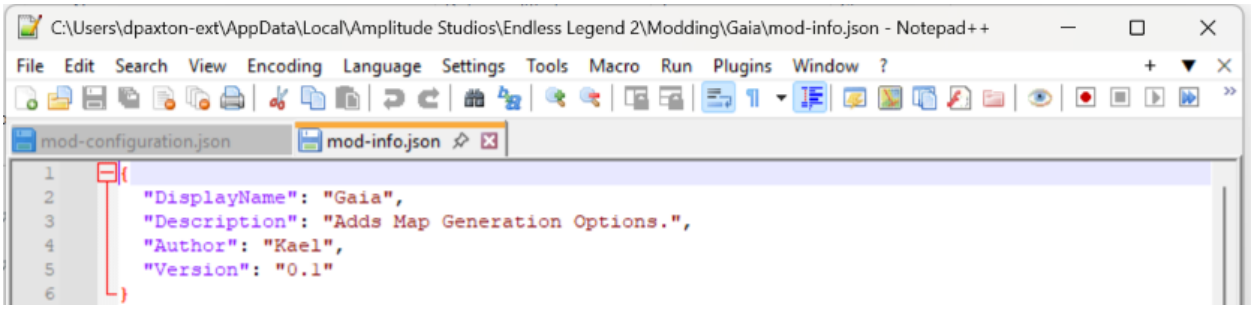


In this example, I am loading 3 local mods, Gaia, BattleStyles and LargerUnits.

3. Create a directory under Modding that matches the name of your mod. In our example here: Gaia.



4. In the Gaia directory, create a mod-info.json file will the mod details.



That’s all you need and you have a Mod. Granted it doesn’t do anything yet. That’s the hard part.

Modifying an Asset

Now we return to that unending (one could say ENDLESS) buffet of game data in the export. If we look in the UnitDefinition_MinorFaction definitions, we find a lot of data, but none of them are the things we most likely want to modify (like the hit points and damage). Instead, we see:

Unit_MinorFaction_Gorog.json


```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Mercury.Data.Simulation.LandUnitDefinition, Amplitude.Mercury.Data,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": ""
  },
  "serializationData": {
    "SerializedFormat": 2,
    "SerializedBytes": [],
    "ReferencedUnityObjects": [],
    "SerializedBytesString": "",
    "Prefab": {
      "instanceID": 0
    },
    "PrefabModificationsReferencedUnityObjects": [],
    "PrefabModifications": [],
    "SerializationNodes": [
      {
        "Name": "GainValues",
        "Entry": 6,
        "Data": ""
      }
    ]
  },
  "IsObsolete": false,
  "Hidden": false,
  "DLCPrerequisite": {
    "DownloadableContent": 0
  },
  "Category": 6,
  "SerializableFamily": "UnitFamily_Nomad",
  "Level": 1,
  "Unicity": 0,
  "CanBeBoughtOut": true,
  "CanBeCanceled": true,
  "IsPartiallyRefund": false,
```

```
"StartWhenQueued": false,
"ProductionCostDefinition": {
  "Type": 1,
  "Constant": -1,
  "RpnDefinitionReference": {
    "serializableElementName": "ProductionCost_Unit_Elite"
  }
},
"MoneyInstantCostDefinition": {
  "Constant": -1,
  "RpnDefinitionReference": {
    "serializableElementName": ""
  }
},
"InfluenceInstantCostDefinition": {
  "Constant": -1,
  "RpnDefinitionReference": {
    "serializableElementName": ""
  }
},
"ResourcePrerequisites": [],
"MinimalPopulationPrerequisite": 0,
"MinimalCityLevelPrerequisite": 0,
"EraPrerequisite": {
  "Operator": 1,
  "EraReference": {
    "serializableElementName": "Era0"
  }
},
"FactionTraitPrerequisites": [],
"ProtectoratePrerequisite": {
  "Operator": 0,
  "MinorFaction": {
    "serializableElementName": "MinorFaction_Gorog"
  }
},
"SettlementStatusPrerequisite": {
  "CampConstraint": 0,
  "CityConstraint": 1
},
"SettlementPropertyPrerequisites": [],
"SettlementApprovalPrerequisite": {
  "Operator": 0,
  "SettlementApprovalDefinitions": []
},
"DistrictCountPrerequisites": [],
"HideInUniquesConstructibles": false,
"AttackSkill": {
  "serializableElementName": ""
},
"UnitClass": {
  "serializableElementName": "UnitClass_Juggernaught"
},
```

```
  "NextEvolutions": [
    {
      "serializableElementName": "Unit_MinorFaction_Gorog_Upgraded"
    }
  ],
  "OwnDescriptorReferences": [
    {
      "serializableElementName": "UnitDescriptor_MinorFaction_Gorog"
    },
    {
      "serializableElementName": "UnitDescriptor_UpkeepLow"
    },
    {
      "serializableElementName": "UnitDescriptor_SpoilOfWarLow"
    },
    {
      "serializableElementName": "UnitDescriptor_ExperienceValue_Medium"
    },
    {
      "serializableElementName": "Tag_Unit_MinorFactionUnit"
    }
  ],
  "OwnAbilityReferences": [
    {
      "serializableElementName": "UnitAbility_InpenetrableMomentum"
    },
    {
      "serializableElementName": "UnitAbility_AlwaysRetaliate"
    }
  ],
  "visualAffinityEraReference": {
    "serializableElementName": "Era0"
  }
}
```


Instead of being the place where the specific stats are set. The Unit Definition is a collection of links (descriptors) for the Unit. This way they can share descriptors with other units if we want.

Some interesting things we see here. This Gorog unit is UnitClass_Juggernaught. It can be upgraded to Unit_MinorFaction_Gorog_Upgraded. It has the UnitAbility_InpenetrableMomentum and UnitAbility_AlwaysRetaliate abilities. And we see a list of descriptors being applied to it.


```
  "OwnDescriptorReferences": [
    {
      "serializableElementName": "UnitDescriptor_MinorFaction_Gorog"
    },
    {
      "serializableElementName": "UnitDescriptor_UpkeepLow"
    },
    {
      "serializableElementName": "UnitDescriptor_SpoilOfWarLow"
```

```
    },
    {
      "serializableElementName": "UnitDescriptor_ExperienceValue_Medium"
    },
    {
      "serializableElementName": "Tag_Unit_MinorFactionUnit"
    }
  ],
```


So, for example, we see that unit upkeep costs are set as a descriptor on units. We can change this to a different Upkeep or remove it entirely. But more interesting is the first one: UnitDescriptor_MinorFaction_Gorog.

Looking at our export, we find that descriptor in the UnitDescriptor_MinorFaction directory.

UnitDescriptor_MinorFaction_Gorog.json


```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Framework.Simulation.Description.Descriptor, Amplitude.Framework,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": ""
  },
  "Effects": [
    {
      "ApplyEffectOnSource": false,
      "Path": {
        "PropertyToFollow": [],
        "Validations": [],
        "specificTargetType": ""
      },
      "PropertyEffects": [
        {
          "Note": "",
          "TargetProperty": "LandLeaderPriority",
          "ToTargetOperation": 0,
          "RpnOperationStack": [],
          "ConstantStack": [
            {
              "RawValue": 3004000
            }
          ],
          "PropertyLocalName": []
        },
        {
          "Note": "",
          "TargetProperty": "HealthPoints",
          "ToTargetOperation": 0,
          "RpnOperationStack": [],
          "ConstantStack": [
            {
              "RawValue": 360000
```

```
        }
      ],
      "PropertyLocalName": []
    },
    {
      "Note": "",
      "TargetProperty": "DamageReductionFlat",
      "ToTargetOperation": 0,
      "RpnOperationStack": [],
      "ConstantStack": [
        {
          "RawValue": 10000
        }
      ],
      "PropertyLocalName": []
    },
    {
      "Note": "",
      "TargetProperty": "DamageMin",
      "ToTargetOperation": 0,
      "RpnOperationStack": [],
      "ConstantStack": [
        {
          "RawValue": 40000
        }
      ],
      "PropertyLocalName": []
    },
    {
      "Note": "",
      "TargetProperty": "DamageMax",
      "ToTargetOperation": 0,
      "RpnOperationStack": [],
      "ConstantStack": [
        {
          "RawValue": 55000
        }
      ],
      "PropertyLocalName": []
    },
    {
      "Note": "",
      "TargetProperty": "LandSpeed",
      "ToTargetOperation": 0,
      "RpnOperationStack": [],
      "ConstantStack": [
        {
          "RawValue": 3000
        }
      ],
      "PropertyLocalName": []
    }
  ],
```

```
        "IgnorePropertyEffectLoopCheck": false
      }
    ],
    "serializableCategory": "",
    "startingType": "Amplitude.Mercury.Simulation.Unit, Amplitude.Mercury.Firstpass"
}
```


Here we see the fun stuff: HealthPoints, DamageReductionFlat, DamageMin, DamageMax, and LandSpeed. Notice that they aren't always called the same thing they are in game, a lot of these attributes have been renamed many times, but the name in data remains the same. Also notice that the values are 1000 larger than they should be. Rather than store all this data with decimals, we multiply everything by 1000 then divide by 1000 when we use it. The programmers tell me this is a good thing. I think we ran low on decimal points or something.

So, if we want to increase the Gorog's Hit Points from 360 to 720 we need to:

1. Copy the UnitDescriptor_MinorFaction_Gorog.json file from the export into our Mod directory.
2. Edit that file in this section, from:


```
"TargetProperty": "HealthPoints",
"ToTargetOperation": 0,
"RpnOperationStack": [],
"ConstantStack": [
  {
    "RawValue": 360000
  }
```


   To:


```
"TargetProperty": "HealthPoints",
"ToTargetOperation": 0,
"RpnOperationStack": [],
"ConstantStack": [
  {
    "RawValue": 720000
  }
```


And that's it. We have made our first mod that does something!

Adding an Asset

Creating a new asset follows a similar process. But to do it we must first copy an existing asset. Let's say we want to add a new faction trait that can be used when making a custom faction, a trait that gives all units increased base damage but reduced Retaliation damage.

We will make the BattleStyles mod, so we need to go through the steps earlier in setting up a mod.

First, we find a custom faction trait to copy. Our teachers lied to us, copying is not only okay, it's the preferred way to get stuff done. In the FactionTraitCustom directory of our exports we find a bunch of traits we can use as a base. Copy one into our Modding/BattleStyles directory. Let's use:

FactionTrait_Custom_Specific02.json


```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Mercury.Data.Simulation.FactionTrait, Amplitude.Mercury.Data,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": ""
  },
  "serializationData": {
    "SerializedFormat": 2,
    "SerializedBytes": [],
    "ReferencedUnityObjects": [],
    "SerializedBytesString": "",
    "Prefab": {
      "instanceID": 0
    },
    "PrefabModificationsReferencedUnityObjects": [],
    "PrefabModifications": [],
    "SerializationNodes": [
      {
        "Name": "SimulationEventEffects",
        "Entry": 7,
        "Data": "0|Amplitude.Mercury.Data.Simulation.SimulationEventEffect[], Amplitude.Mercury.Data"
      },
      {
        "Name": "",
        "Entry": 12,
        "Data": "1"
      },
      {
        "Name": "",
        "Entry": 7,
        "Data": "1|Amplitude.Mercury.Data.Simulation.SimulationEventEffect_ApplyDescriptor,
Amplitude.Mercury.Data"
      },
      {
        "Name": "SimulationEffectDescriptionOverride",
```

```
    "Entry": 6,
    "Data": ""
  },
  {
    "Name": "UIMapperOverride",
    "Entry": 1,
    "Data": ""
  },
  {
    "Name": "Hidden",
    "Entry": 5,
    "Data": "false"
  },
  {
    "Name": "GainValues",
    "Entry": 6,
    "Data": ""
  },
  {
    "Name": "TargetID",
    "Entry": 1,
    "Data": "Empire"
  },
  {
    "Name": "Descriptor",
    "Entry": 7,
    "Data": "Amplitude.Framework.DatatableElementReference, Amplitude.Framework"
  },
  {
    "Name": "serializableElementName",
    "Entry": 1,
    "Data": "Effect_Custom_Specific02"
  },
  {
    "Name": "",
    "Entry": 8,
    "Data": ""
  },
  {
    "Name": "",
    "Entry": 8,
    "Data": ""
  },
  {
    "Name": "",
    "Entry": 13,
```

```
            "Data": ""
          },
          {
            "Name": "",
            "Entry": 8,
            "Data": ""
          }
        ]
    },
    "IsObsolete": false,
    "DLCPrerequisite": {
      "DownloadableContent": 0
    },
    "Cost": 10,
    "ForbiddenFactionTrait": [],
    "MandatoryFactionAffinity": {
      "serializableElementName": ""
    },
    "BoundFactionTrait": [],
    "IsAvailableForCustomFaction": true,
    "Category": {
      "serializableElementName": "FactionTraitCategory_Defense"
    }
}
```


Rename the file. Since this is a new asset, it cannot have the same name as an existing asset. Instead, we will rename it to FactionTrait_Custom_Rage.json.

Set the SourceElementName to the original asset. This is because every Add is still a Modify. But with the additional step of pointing to an asset we want to copy. The json files don't have enough data to make an entirely new asset, so instead we use the copied asset as a base, then we tell the game to apply all the the specified changes In our case we start with the header of:


```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Mercury.Data.Simulation.FactionTrait, Amplitude.Mercury.Data,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": ""
  },
```


And change it to:


```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
```

```
    "AssetTypeName": "Amplitude.Mercury.Data.Simulation.FactionTrait, Amplitude.Mercury.Data,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": "FactionTrait_Custom_Specific02"
  },
```

Now we have a Rage Custom faction trait that is exactly like FactionTrait_Custom_Specific02. If we look at the original trait, we see that all it's really doing is applying an effect, specifically Effect_Custom_Specific02. Let's change that to Effect_Custom_Rage (which doesn't exist yet).

Now, we need to create that new effect using the same process. We copy Effect_Custom_Specific02 (which we find in the FactionTraitCustomDescriptor directory) into our mod directory and rename it Effect_Custom_Rage and being sure to set the SourceElementName to the asset we are copying (Effect_Custom_Specific02).

Then we modify it to the effect we want:

Effect_Custom_Rage.json

```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Framework.Simulation.Description.Descriptor, Amplitude.Framework,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": "Effect_Custom_Specific02"
  },
  "Effects": [
    {
      "ApplyEffectOnSource": false,
      "Path": {
        "PropertyToFollow": [
          "Armies",
          "Units"
        ],
        "Validations": [],
        "specificTargetType": ""
      },
      "PropertyEffects": [
        {
          "Note": "",
          "TargetProperty": "RetaliateDamageModifier",
          "ToTargetOperation": 4,
          "RpnOperationStack": [],
          "ConstantStack": [
            {
              "RawValue": -500
```

```
            }
          ],
          "PropertyLocalName": []
        },
        {
          "Note": "",
          "TargetProperty": "DamageBonusModifier",
          "ToTargetOperation": 4,
          "RpnOperationStack": [],
          "ConstantStack": [
            {
              "RawValue": 100
            }
          ],
          "PropertyLocalName": []
        }
      ],
      "IgnorePropertyEffectLoopCheck": false
    }
  ],
  "serializableCategory": "",
  "startingType": "Amplitude.Mercury.Simulation.MajorEmpire, Amplitude.Mercury.Firstpass"
}
```


There is a lot happening here. Let's go through the important changes.

1. The SourceElementName is correctly set to the asset we copied.
2. There is an effect that is being applied to Armies and Units. The best way to know how to do this is to look for assets that are already doing things similar to what you want.
3. There are two PropertyEffects being applied. The first gives -50% to RetaliateDamageModifier and the 2nd gives +30% to DamageBonusModifer. To figure out the correct TargetProperty name checkout other assets. A "4" in ToTargetOperation" means that the RawValue is being applied as a percent. There is a table in the appendix of this document that lists all the ToTargetOperation values.

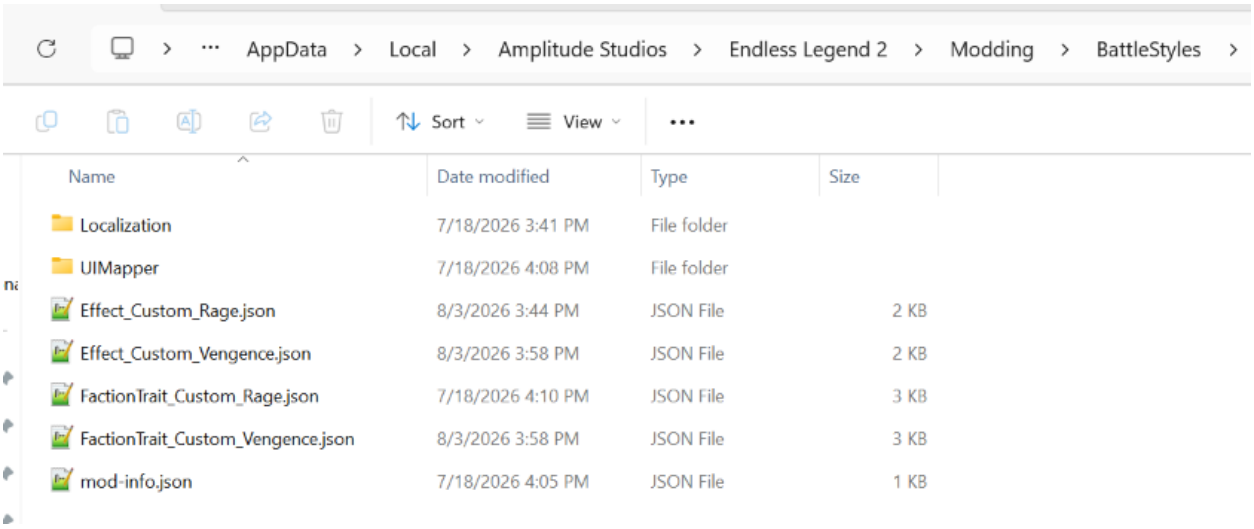
And now we have a new custom faction trait that adds +10% to base damage and -50% to Retaliation Damage.

But, we aren't done.

UIMappers

When we add new assets that are displayed in game we also need to have a UIMapper to connect them to the UI components of the game. Because we prefer to have different assets for different functions (as evidenced by the many links between objects and the 30,000 data elements in the export) this is a separate asset.

The way that an asset knows which UIMapper is correct is that they have the exact same name. This is a little odd, but at least it saves us from having to assign a field in an asset pointing to the UIMapper. But we need to make sure that when we make a new UIMapper for an asset it has **exactly** the same filename as the asset. Since you can't have 2 files with exactly the same name in the same directory we will need to create a subdirectory to put one of them in. I prefer making a UIMapper directory in our mod's directory to store them all, but it's up to you. It doesn't really matter what the directory name is, you can organize as you prefer. It ends up looking like this:



Just like when creating any new assets we first copy an existing asset. In this case the FactionTrait_Custom_Specific02.json from the FactionTraitCustomUIMappers directory (not to be confused with the file of same name from the FactionTraitCustom directory). Then we rename it to FactionTrait_Custom_Rage.json (the same name as our new asset).

FactionTrait_Custom_Rage.json (UIMapper)

```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Mercury.UI.FactionTraitUIMapper, Amplitude.Mercury.Data,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": "FactionTrait_Custom_Specific02"
  },
```

```
"RawTitle": "%FactionTrait_Custom_RageTitle",
"Description": "%FactionTrait_Custom_RageDescription",
"Lore": "",
"Images": [
  {
    "Value": {
      "guid": {
        "a": -1777478323,
        "b": 1242587273,
        "c": 1038124711,
        "d": -1259204341
      },
      "virtualTextureKeyHash": 0
    },
    "serializableKey": "Picto"
  },
  {
    "Value": {
      "guid": {
        "a": 1255629589,
        "b": 1341626172,
        "c": -175111507,
        "d": -1662543613
      },
      "virtualTextureKeyHash": 0
    },
    "serializableKey": "Small"
  }
],
"UIColor": {
  "paletteGuid": {
    "a": 0,
    "b": 0,
    "c": 0,
    "d": 0
  },
  "colorGuid": {
    "a": 0,
    "b": 0,
    "c": 0,
    "d": 0
  },
  "freeColor": {
    "r": 1.0,
    "g": 1.0,
    "b": 1.0,
```

```
        "a": 1.0
      },
      "modifierHolder": {
        "isEnabled": false,
        "alpha": 0.0
      }
    },
    "Symbol": "",
    "OptionalTag": "",
    "Shown": -1,
    "HideSimEventEffects": false,
    "LocalizationLines": []
}
```

The important details here are:

1. We correctly set the SourceElementName to the asset we copied.
2. We set the RawTitle to a localization string: %FactionTrait_Custom_RageTitle.
3. We set the Description to a localization string: %FactionTrait_Custom_RageDescription.

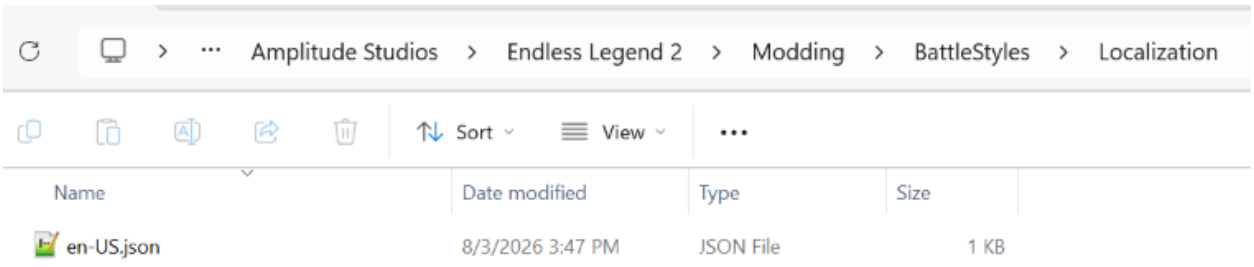
Which is great, but now we need to add the new localization strings.

NOTE: When we copy an asset we need to look through it and remove the things we don't want out new asset to have. In the above example the original asset had an entry in "LocalizationLines". I removed that so that it would use the Localization we are about to add, but until this is done "old" data will continue to show up in our new assets. This doesn't apply only to localization, but anything that got brought over in the copy.

NOTE #2: Even though there is a lot that can be done with the modding in ENDLESS Legend 2, don't assume that it will be automatically feedbacked. In this example we modify Retaliation damage, but this won't show up automatically in the tooltips.

Adding Text

This is the last part, and it's super simple. We need to add new strings for our mod. To do that, create a Localization directory and create a new file named according to the language we want to add. IN my case I want to add English, so I add en-US.json.



That file is just a simple json file with the KeyName:String. For the BattleStyles mod I have:

en-US.json

```
{
	"%FactionTrait_Custom_RageTitle":"Rage",
	"%FactionTrait_Custom_RageDescription":"+30% Base Damage but -50% Retaliation damage for all Units and Heroes.\r\n\r\nProvided by: <c=FF0000>Battle Styles Mod</c>",
	"%FactionTrait_Custom_VengenceTitle":"Vengence",
	"%FactionTrait_Custom_VengenceDescription":"-25% Base Damage but +100% Retaliation damage for all Units and Heroes.\r\n\r\nProvided by: <c=FF0000>Battle Styles Mod</c>"
}
```

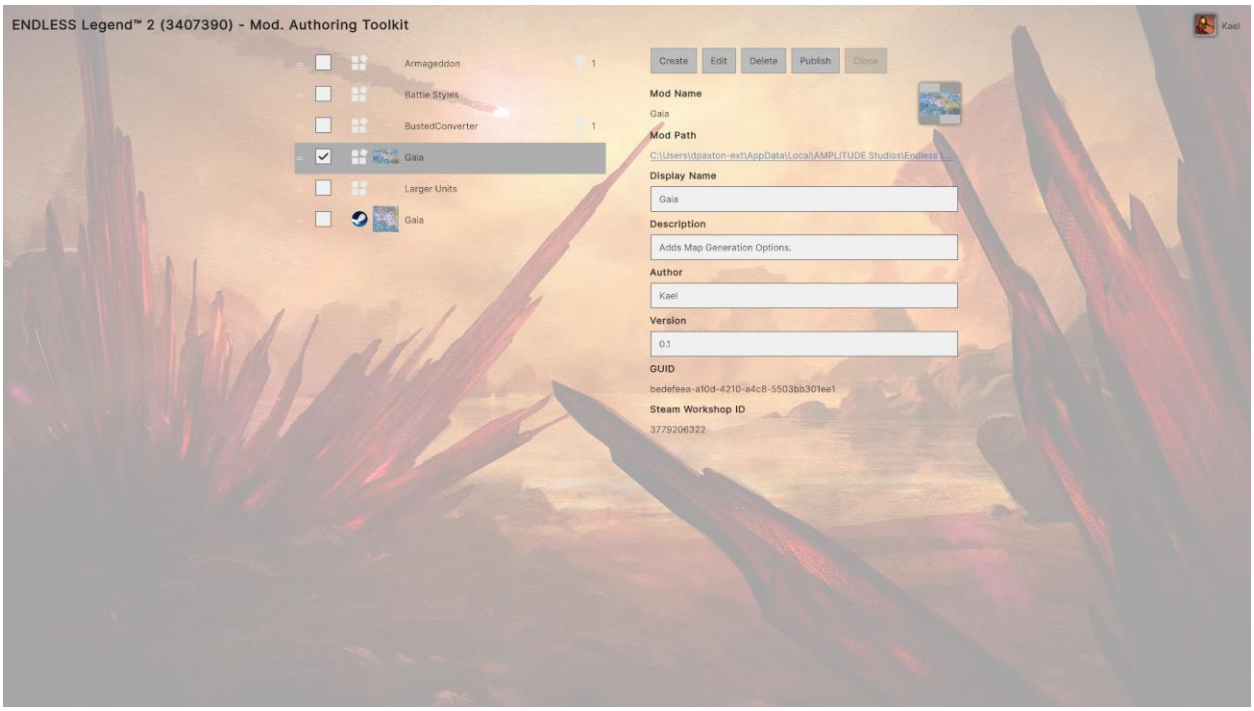
Notice the control characters for newline breaks (\r\n) and the color codes (<c=FF0000> it pure red and </c> ends the color change).

And that is it, you are ready to play with your new assets.

Unsupported Languages

If someone plays your mod in a language that you don't have language files for, it will failback to the English version of the string. This is to keep the mod from just displaying a missing string if someone plays in Russian (for example) and you didn't include Russian localization. Instead, they will at least see the English strings.

How to Upload your Mod



This part is easy as well. We have provided a separate tool that will see your local mods and allow you to edit their details and upload them. It will read in the data in your modinfo file, but you can also edit the information here (including the image) by pressing the Edit button.

Steam has a limit on the size of that image, it can't be larger than 1 MB or the upload will fail. It works as a 1920x1080 jpg, but pngs that are much larger are probably won't work.

You can find the Toolkit in your game directory at "..\ENDLESS Legend 2\Public\Modding\Toolkit\Mod. Authoring Toolkit.exe".

Sample Mods

All these mods are available in the Steam Workshop so you are welcome to download them, play with them or pick through them for any details that help you create your own mods.

Gaia

World Generation Mod

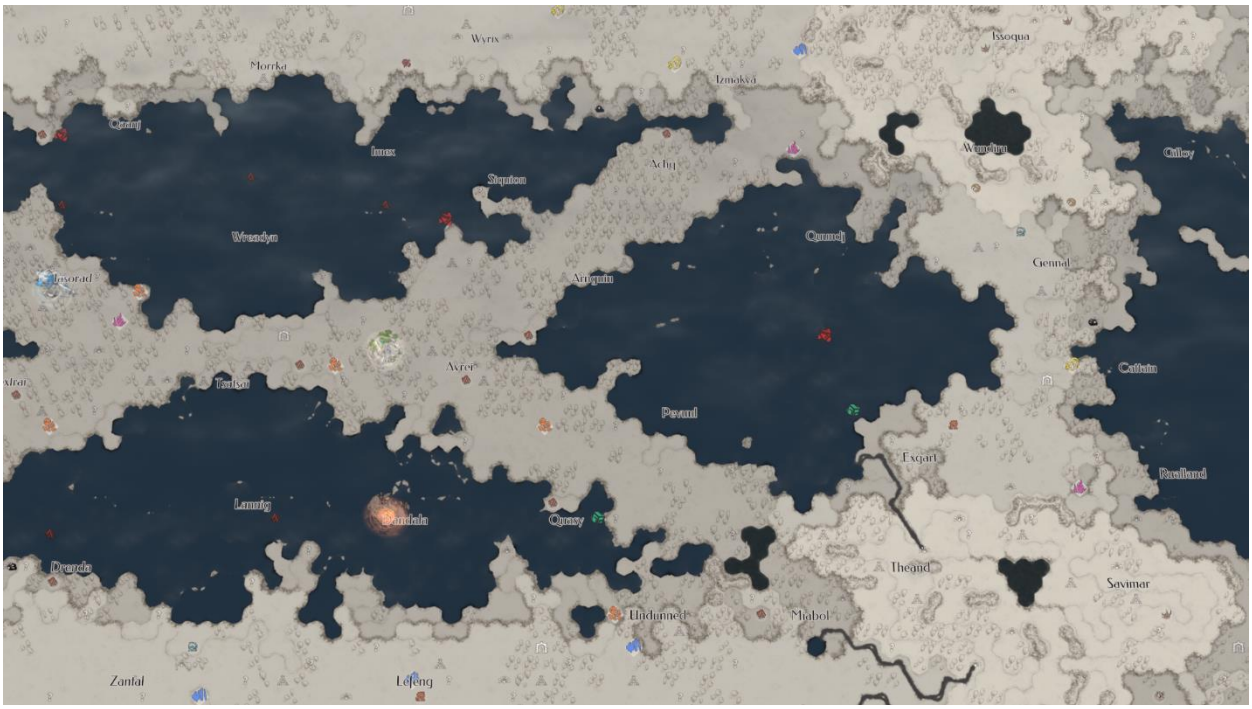
Gaia allows you to have more variety in the games you make and adds the following options to game setup:

New World Sizes

- **Gaia (small)**- This World Size is 50% taller and wider than the Huge map with Small territories, so there are tons of territories to conquer.
- **Gaia (large)**- This is the same 50% taller and wider world size, but it also has larger territories so the territory count remains about the same, but they are also huge.

New World Settings

- **Lake Odds- Wetlands**. Lakes are even more likely than at the highest base setting. This is good for the larger Gaia maps.
- **Lake Size- Yo'Mamma**. Lakes can be up to 64 tiles in size. Another great adjustment for the larger Gaia maps, but you might not want to pair this with Wetlands lake odds.
- **Ridges and Cliffs- Flatlands**. Ridges will be incredibly infrequent. A terrible map for the Tahuk.
- **Ridges and Cliffs- Mountainous.** The world is covered with long chains of ridges, making for a lot of chokepoints, valleys and little room for cities to spread.



Larger Units

Increases the size of all units in the game by about 50%. This shows off modifications that can be done in the presentation layer of the game as well as just generally creating some monstrous units running around the map. Note this changes the scaling of the units on the world map, in battle and in the equipment screen.

This mod is super simple. I copied every entry from the PresentationPawnDefinition directories into the mod. These are the definitions for all the unit models in the game.

Each of these files has scale for various components, but the main one we are interested in is:

PresenationPawn_Aspect_Ranged.json (snippet)

```
"MaxHeight": 2.200000047683716,
"Scale": 1.0,
"CapsuleLength": 0.0,
"CapsuleRadius": 0.4000000059604645,
"AnimationCapabilities": 6,
"Death01AnimationParameters": {
  "DeathMaxVerticalPushImpulse": 0.0,
  "DeathPushDistance": {
    "x": 0.0,
    "y": 0.0
  }
},
```

If we change that Scale from 1.0 to 1.5 we get a 50% larger unit. This mod goes through all the PresenationPawn definitions and increase the scale values.

Larger Armies

This mod increases the maximum number of units per army from 6 to 8. It adds 2 new technologies, one to the 3rd Era and one to the 5th Era. Each of those technologies unlocks a new Army Slot.

With Reinforcements it means you can now have 16 vs 16 unit battles!

This screenshot was made with both Larger Units and Larger Armies mods enabled as larger Kin and Last Lords armies face off.

This is a more complex mod and I recommend anyone interested download it and check out all the specifics. But overall it does 3 things:

Modifies the base game definitions

Tag_Empire_Major.json is a great file because all the major factions inherit it. Base factions and custom factions. So any change made here will be inherited by all players. Here we make one addition:

Tag_Empire_Major.json (addition)

```
{
        "Note": "",
        "TargetProperty": "MaxArmyMaximumSize",
        "ToTargetOperation": 0,
        "RpnOperationStack": [],
        "ConstantStack": [
          {
            "RawValue": 2000
          }
        ],
        "PropertyLocalName": []
}
```

This Adds (“ToTargetOperation”: 0) 2 (“RawValue”: 2000) to our Maximum Army Size (“TargetProperty”: “MaxArmyMaximumSize”).

The Two technologies are added using the process for adding a new technology covered earlier in this doc as they apply the new Descriptor “Effect_ArmyMaximumSize_00” (or 01 for the second technology).

And lastly we add those two new descriptors

Effect_ArmyMaximumSize_00.json

```
{
  "AMP_SERIALIZATION_HEADER": {
    "Version": 0,
    "AssetTypeName": "Amplitude.Framework.Simulation.Description.Descriptor, Amplitude.Framework,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null",
    "SourceElementName": "Descriptor_Era2_UnitSlotUnlock"
  },
  "Effects": [
    {
```

```
      "ApplyEffectOnSource": false,
      "Path": {
        "PropertyToFollow": [],
        "Validations": [],
        "specificTargetType": ""
      },
      "PropertyEffects": [
        {
          "Note": "",
          "TargetProperty": "ArmyMaximumSize",
          "ToTargetOperation": 0,
          "RpnOperationStack": [],
          "ConstantStack": [
            {
              "RawValue": 1000
            }
          ],
          "PropertyLocalName": []
        }
      ],
      "IgnorePropertyEffectLoopCheck": false
    }
  ],
  "serializableCategory": "",
  "startingType": "Amplitude.Mercury.Simulation.MajorEmpire, Amplitude.Mercury.Firstpass"
}
```


This new Descriptor uses Descriptor_Era2_UnitSlotUnlock as a source and changes nothing (since that descriptor does the same thing). You may wonder why the mod needs a new descriptor at all (let alone 2 identical ones). Can't the new techs just point to the existing Descriptor that already unlocks a new army slot?

But that doesn't not work because an asset cannot have the same descriptor more than once. If we have 2 triggers that give the same Descriptor, the 2nd one is ignored. So, we need a new descriptor if we want these effects to stack.

Troubleshooting Errors

Modding Toolkit and ENDLESS Legend 2 Conflicts

The Modding Toolkit uses the same Application ID as EL2. This is required so that Modding Toolkit can use your steam account to upload mods. But it also means that (from Steam's perspective) they are the same application. So, if you have the Toolkit running, Steam will think that EL2 is already running and not allow you to start it again. Shut down the Toolkit before starting EL2.

Handling Errors

We try to do some data validation when we load mods, and provide some information about what went wrong. But, there are too many things to predict and in some cases the error message won't be very helpful. In those cases there are a few tricks to help isolate the issue:

1. Test changes at each step. Make small changes and load the game to make sure they work before going on to the next part so you know exactly what changed when the issue starts.
2. Backup your mod when you have a working build. If you have a mod you like and you want to make a big change it's a good time to make a backup copy just in case. Or if

you are starting your day it's a good idea to make a backup. Or really, just about any time is a good time to make a backup. Disk space is cheap, breaking the mod you spent days working on without a backup is expensive.

3. If an error happens on game start it can be due to missing or not well formed asset. Errors can be seen in diagnostics file and can help to understand what is wrong. Search in '%LOCALAPPDATA%\Amplitude Studios\Endless Legend 2\Temporary Files\Diagnostics *.html' for these errors. For example, '[UI] Could not find UIMapper 'FactionTrait_Custom_Pawa' or something similar and related to your mod.

Appendix - Data Definitions

Mukag?

You will see lots of references to the Mukag faction in data. This was our original name for the Tahuks. Flavor names change frequently and we don't go back and rename all the data every time (because they will just change again), but it does mean that you need to keep in mind that the game name for things, may be very different than the data name.

ToTargetOperation

Many time you will see effects being applied to various stats and resources with ToTargetOperation and a number. It could be giving +10, -10, +10%, etc. This is what those numbers decode to:

0 = Add
1 = Sub
2 = Mult
3 = Div
4 = Percent
5 = Pow
6 = Max

Attributes

Much as we have renamed factions and units during development, the names of attributes have also gone through a lot of changes. If you want to modify these attributes in data you will need to use the data names for them.

Strength = Might
Dexterity = Determination
Constitution = Resilience
Intellect = Intuition