Anniversary Edition Framework

Introduction

The Anniversary Edition Framework was created to fix a problem in Oni's modding community. Before its creation, community member "Neo" created a tool named Onisplit, allowing Oni's proprietary data files to be split into easily manageable chunks, each containing one portion of the game data that could be edited independently from the others. Furthermore, many types of data could be converted to human readable XML files, for further ease of editing. This was a much better solution than previous attempts at making modding much accessible, as Oni's data consisted of thousands of files merged together into thirty different data data files, merged into thirty large files (45 in the Mac version). Being able to edit them separately and being able to use your OS's familiar filesystem tools to find the correct files you need made things much easier. This also made sharing changes much easier. Filesizes of changes are much smaller, changes can be merged together, and the legal problem of sending entire levels of the game across the internet is virtually eliminated.

However, there were still problems.

1. Much of Oni's data is repeated across levels. It is terribly inefficient, both in hard drive space and for sending changed files to other players for things to be this way.
2. The only way to install changes was to split up Oni's data, add your files in, then recombine the data. This could only be done through the command line or through a batch\shell script, leaving it very unfriendly to users who aren't tech savvy.
3. Users aren't always very smart. They will alter the original data without creating a backup, leaving them out of luck if they want to uninstall, a modification is causing trouble, or if they end up somehow scrambling files up.
4. There was no easy way to uninstall modifications.

The first attempt at solving these problems was made by community member "Geyser". He created the original Anniversary Edition, using batch scripts to move files around, eliminating duplicates and installing a preset selection of mods. This was a great start, but lead to two more problems.

5. Not everybody was happy with Geyser's selection of modifications. In fact, there is no combination of modifications that will ever make everyone happy. The best he could do was make the majority mostly happy.
6. It was made in complicated shell\batch scripting. If any portion of the process needed changing, you had to get two experts together to fix it, one for each OS.

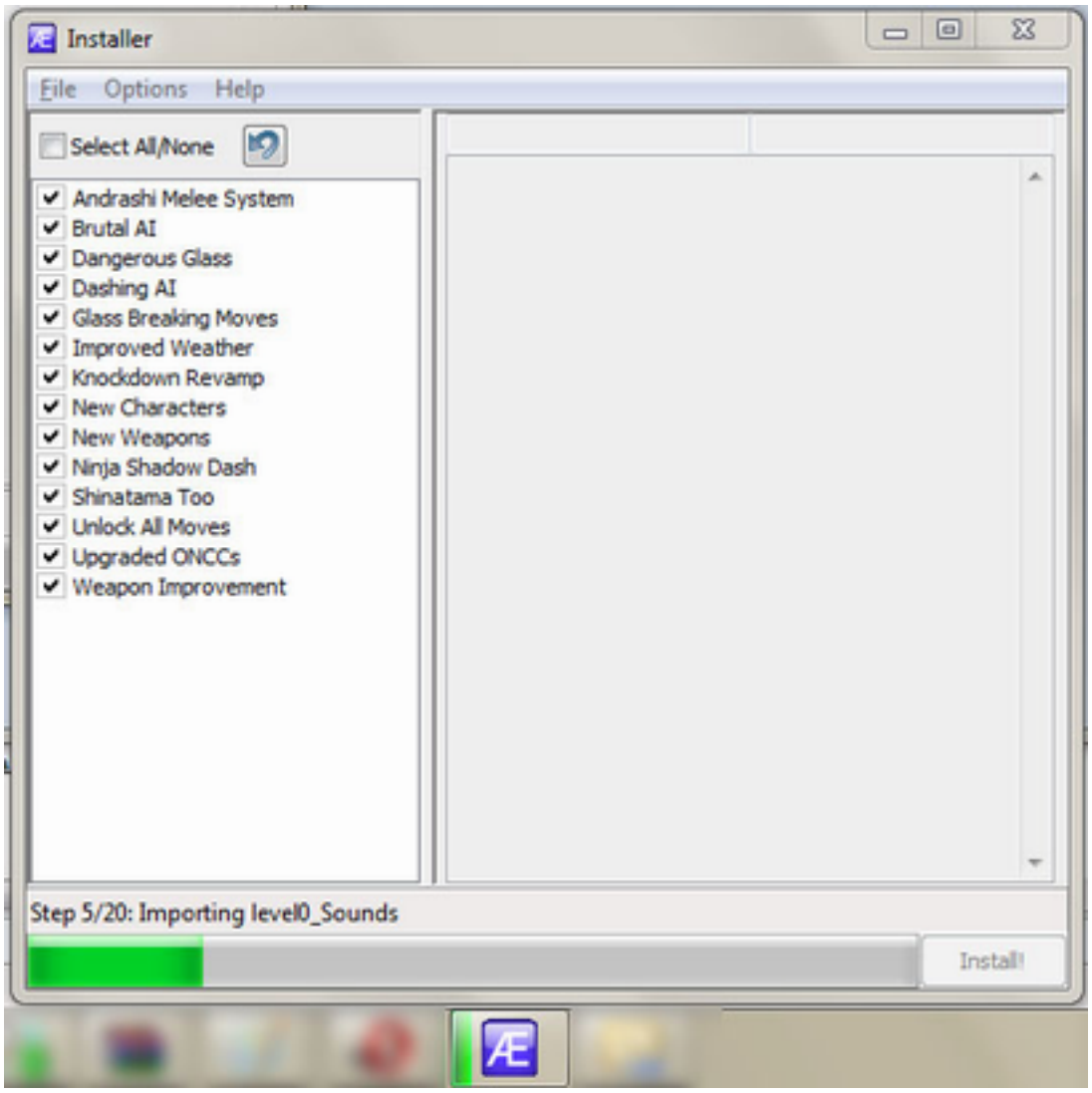
And so, in the winter of 2008 work was started on the Anniversary Edition Installer and Framework.

Overview

The Framework consists of two major parts: the Installer and packages.

The Installer

The Installer contains both the GUI and the underlying file operations code. Here is a screenshot:



The Installer was designed to be as user friendly as possible. The first time the Installer is run, it asks you to run a one time process of “Globalization”. This does several things, including copying all of Oni’s data to the subfolder Anniversary Edition is installed in, removing duplicate files, and moving non level specific files to the folder containing “global” data. This process typically takes no longer than 10 minutes. After that, the user is free to select any modifications he likes using the modification pane (on the left) and

click the Install button. Clicking on a mod's name will bring up a description in the right pane. Installation usually takes less than one minute; once the installation is finished, Oni: Anniversary Edition can be played. The Installer is coded in C++ using Boost for file operations and wxWidgets for the GUI. It compiles on both Windows and OSX.

Packages

Packages are used to store modifications in Oni. They consist of a configuration file and several binary and\or script files. They are located in Edition\install\packages and are loaded when the Installer starts.

Here is the structure of a typical package.

- 00000PackageName\
  - Mod_Info.cfg
  - bsl\
    - tctf_ii
      - script_file.bsl
      - another_script_file.bsl
  - oni\
    - level0_Final
      - global_oni_file.oni
      - foo.oni
    - level2_Final
      - local_oni_file.oni
      - another_foo.oni
      - bar.oni

The package name is used for organizational purposes. Packages which start with a higher number have higher priority when there is a conflict between packages with the same files inside.

Structure

Mod_Info.cfg contains information on the enclosed package. Here is a sample configuration file:

```
AEInstallVersion -> 1.1
NameOfMod -> Improved Weapons
ModVersion -> 1.1
Category -> AE: Weapons
Creator -> Geyser/Loser/Iritscen/Gumby
HasOnis -> Yes
Readme -> Improves the appearence and mechanics of existing Oni
weapons. \n \n Currently contains…
```

A field has the format "Key -> Value".

A listing of keys and the effects associated with changing each value:

AEInstallVersion: If this is set to be higher than the version of the Installer you are currently using, it will be ignored.

NameOfMod: The name of the mod, shown in the Installer.

ModVersion: Used by update code to determine whether or not a mod is up to date.

Category: Currently unused

HasOnis: Tells the Installer to read from the oni\ folder in the package for binary data.

HasBSL: Tells the Installer to read from the bsl\ folder in the package for level script data.

Readme: The description shown when you click on the name of the mod in the Installer. Add "\n" to add a newline inside the description.

Creator: The name(s) the Installer credits for making the mod.

The oni\ folder

Inside the oni\ there are subfolders corresponding to Oni's level named "level0_Final, level1_Final, etc". Only one folder is needed, but you can have one for each level and even include extra folders for new level. Each folder contains binary data for the Installer to combine with Oni's data.

The bsl\ folder

Inside the bsl\ folder there are subfolders named after the script folders Oni uses for storing level scripts. These either replace or add on to the scripts Oni has, depending on if you use a value of "Yes" or "Addon" for the "HasBSL" key.

The installation process in depth

Oni's data format

Oni's data consists of dat, raw, and sep files. Dat files contain fixed width data. Raw files are linked to by dat files. They typically contain unstructured files such as textures, sounds, and models, or other binary data that cannot fit into a normal C structure. Sep files serve an identical purpose to raw files and are used in conjunction with raw files on the Mac build of Oni. Each dat file links to one raw(/sep) file of identical name (level1_Final.dat links to level1_Final.raw). Each level gets one set of files and one set

also exists to be accessed by all levels and the start screen. This special set of files is referred to as “level 0”.

Onisplit

Inside each dat file exists hundreds of what were once files (let’s call them “instances”). These instances are loaded up into Oni’s memory at startup and can contain everything from character data to animation information. Onisplit takes the dat\raw(\sep) for a level and splits each instance into an individual file, containing both the dat portion and the raw portion, if necessary. A more complete explanation of Onisplit can be found at http://wiki.oni2.net/OniSplit
.

Preinstallation

Before actually “installing” Anniversary Edition, the user must extract the contents of the .zip file containing AE directly to their Oni folder. Failing that, a user can extract to another folder and move the Edition folder inside Oni\. Unfortunately, extracting to the wrong folder is the number one error behind failed Anniversary Edition installation. Future Anniversary Edition releases will include some method of checking for a proper Oni install in the parent folder.

Here is the file structure of a properly extracted Edition installation, with new files in bold:

- Oni\
  - GameDataFolder\
    - IGMD\
      - *Script folders*
    - *data files*
  - **Edition**
    - **install**
      - **packages**
        - ***package folders***
      - **AEInstaller.exe (AEInstaller.app)**
      - **Onisplit.exe**
      - **onisplit_gui.exe (Onisplit Gui.app)**
      - **ospgui_lang.ini**
    - **AE Read-Me.htm**
    - **binkw32.dll (PC only)**
    - **Oni.exe (Oni.app)**
    - **run_wind.bat (.sh)**
    - **run_full.bat (.sh)**
  - Oni.exe (Oni.app)
  - binkw32.dll (PC only)

- persist.dat

Globalization

As mentioned above, one file group is special in that it can be accessed no matter what level is loaded in the game, or even if you aren't in a level at all. The instances in these files are referred to as "globalized". Unfortunately many important instances are **not** globalized. An example: in an unmodded copy of Oni, you cannot play with all characters in all levels. Instead of being in level 0, these instances are copied across to whichever levels they are needed. This creates both a file size problem (a large portion of Oni's instances are duplicated) and a logistics problem for potential modders (having to copy modifications to a dozen different levels is neither fun nor a great way to keep your work consistent between levels). Part of the original purpose of the Anniversary Edition was to fix this. For your enjoyment, an overview of the Installer's globalization process:

1. Run an Onisplit command to extract all data files from Oni\GameDataFolder to Oni\Edition\GameDataFolder\level#_Final (# changing with the level)
2. Move all files that need to be globalized from level#_Final\ to level0_Final. If one already exists there, delete the extra
3. Run an Onisplit command to recombine all the files in each level# folder into a single .dat\raw(\sep) group for each level outputted to the folders Oni\Edition\install\VanillaDats\level#Final\
4. Rename each .dat file to a .oni file. This tricks Onisplit to treating them as (very large) instance files that can be used as input. Reading in one large file with the links between instances already made is much quicker than reading in a few hundred unlinked instances.
5. Copy the savegame file (persist.dat) from Oni\ to Edition\
6. Pass over all the files in Edition\ to remove any readonly properties that zip extraction programs may have assigned.

That's all that needs to be done. While there are a few workarounds for Onisplit bugs in the process that aren't included, they are irrelevant to explaining how globalization works.

You may have noticed that this leaves behind extra folders in GameDataFolder. These are used for modding reference, but can be deleted to save space.

Installation - Data

Once globalization has finished and the user has given the ok to do so, installation can begin. First all old dat\raw\sep files are deleted from Edition\GameDataFolder. Next, an Onisplit command is run for each level. Let's use level 1 as an example:

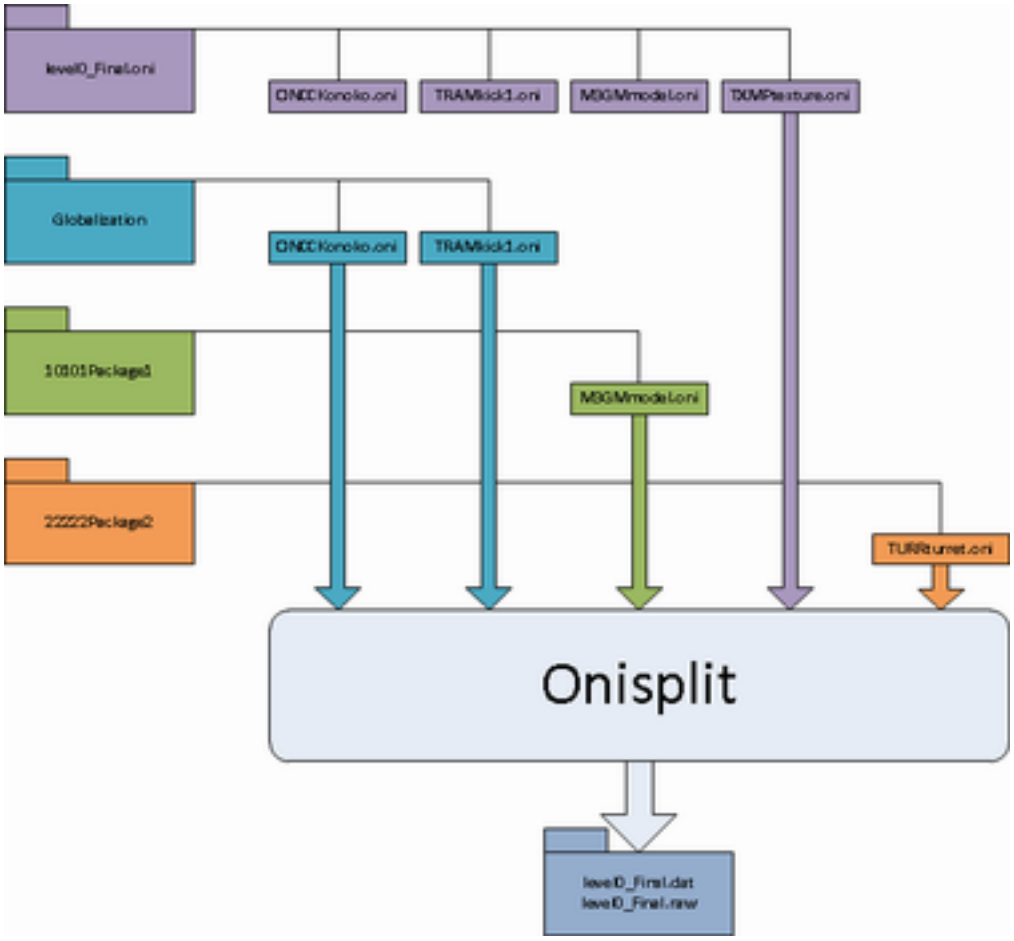
Onisplit –import:(*sep/nosep*) VanillaDats\level1_Final packages\Globalize\oni\level1_Final packages*packagename1*\oni\level1_Final packages*packagename2*\oni\level1_Final … ..\GameDataFolder\level1_Final.dat

It's a bit much at first, so let's break this command down.

–import:sep and **–import:nosep** and used to tell Onisplit to convert a batch of .oni files to a dat\raw(\sep) group. The Installer chooses the correct version between the two at compile time, depending on which OS it is compiled for.

Following the command is a list of folders. The first of these folders is **VanillaDats\level1_Final**. This folder contains unmodified data files from the game and was created in the globalization step. The next of these folders is **packages\Globalization**. This package is invisible to users; it contains files that are either bugfixes or preparations that make new modifications for Oni easier. These are mandatory to install, but uncontroversial in nature. After this is a list of folders corresponding to the packages the user has chosen to install. Last is **..\GameDataFolder\level1_Final.dat**, which is the destination of the newly created dat\raw(\sep).

When this command is run, Onisplit draws up a list of .oni files from those folders, starting with the first folder in the list. If a file with the same name is found in a package later on the list of folders, it replaces the original one in the list of .oni files. After each folder is scanned, the .oni files are loaded into memory, separated into data and raw parts, and linked together into one large instance file. Here is a visualization of the process:



Installation – Scripts

Finally it is time to copy the level scripts. First, all scripts are deleted from Oni\Edition\IGMD\GameDataFolder\. After this, scripts are copied from packages having a “Yes” value for the HasBSL key. If two packages use the same script folder, the one with the higher number gets ownership. The original game scripts count as a “package” with the lowest number possible, just as “Globalize” does for data. Lastly, scripts having an “Addon” value for HasBSL are copied somewhat blindly into whichever script folders they request. These type of script modifications are meant to work with existing scripts rather than replace them.

Installation – Wrap-up

At this point the installation is done and the user is free to start playing Anniversary Edition. The file structure mirrors the structure of the original game. At any time he can come back and change which modifications he would like to use with Oni.

Conclusion

The most important question any software developer needs to ask about his work is this: "Does my program actually solve the problems it was made to fix?" Let's check to see if the problems are solved.

1. Repeated data – solved by globalization
2. Unfriendly install process – solved by automating the process and adding a GUI on top
3. Original game data being modified – solved by moving any modifications into a subfolder
4. Uninstallation woes – solved by allowing any modification to be removed with the click of a button
5. Users wanting different things - solved by giving them the choice to install any package they like
6. Ugly shell\batch scripting – Solved. C++ is much easier to understand and modify than the monstrosities we had before.

I'd have to say mission accomplished.