0. Overview

This launcher refresh (internally named “Kachemak”) is intended to massively expand and modernize the TF2Classic launcher, and more broadly, the user process of installing the game, and the developer process of creating and deploying updates.

For platform support, we will fully support Windows and Linux, including with the GUI. No features will be exclusive. Portability has been a primary focus of the application’s design.

For updates, it reduces the downloaded size of patches to a tiny fraction of what is currently distributed, speeding up the process by several factors and lessening server burden.

For installation of the whole game (now done through the launcher exclusively rather than by downloading and manually extracting the game’s archive), the process is effectively simplified and automated, such that the relevant folders are autodetected, symlinks are created if the user wishes to install to an external drive, and everything is based on simple prompts.

For downloads, Kachemak is incredibly **speedy** (using multiple mirrors simultaneously), **robust** (can recover from any kind of interruption with zero data loss; any corruption can be autorepaired), **flexible** (users can configure support for throttling, proxies, and more), and **reliable** (there is no single-point-of-failure).

Kachemak aims to not reinvent the wheel, but to make clever use of mature external utilities for extremely advanced functionality. While it has a very low burden of development and maintenance, it is far more than the sum of its parts. It is also a free-and-open-source project, with plans to be licensed under the GPLv3 and have community-focused development on GitLab.

It is also a project that can help the ecosystem at-large. It would be trivial to tweak and rebrand it for other projects, such as Open Fortress. By solving the problems that are otherwise intrinsic to using deltas to update VPKs (which may change in unexpected ways between game builds), it is a solution that can hopefully empower other Source projects as well.

1. Updating

1.1 Deltas

With Kachemak, deltas will be the primary method of updating TF2C. In preparation for an update, a selection of the largest folders from two builds of the game will be tarred, with xdelta3 being used to generate binary delta files that contain the differences between the tars generated from the different game versions. In a similar manner, users will tar their local directories, and then download a set of delta files which would be applied to their local tars, and then unpacked again to overwrite their local files and bring them up-to-date with the latest version of the game.

As compared to shipping LZMA-compressed patch archives containing all modified files, this takes up only 15% of the size, and applies much more quickly. As compared to the Deflate-compressed archives the current updater uses, this is less than 10% of the size. Even as assets may move between VPK files, it can track them and move the data instead of redownloading it, meaning that trivial patches are trivial sizes, and apply in only a couple minutes. This can substantially lower the burden on the servers hosting the files, as well as speed up the updating process by many factors, among other benefits.

The purpose of deltas is to handle updating the sections of the game that take up the largest amount of space, such as the VPKs, maps, media, and executables. Smaller changes, such as changes to the /cfg/ folder or any top-level files, will be handled more directly by the updater. On a per-upgrade process, the installer will need to access information over the web regarding smaller files that have been added, removed, or changed, and then handle these accordingly. A list of modified files between major builds of the game can be generated with `diff -qr` , allowing automation of this process.

We cannot efficiently delta files individually, or rely on directory-wide delta solutions (which usually are just a nice wrapper that still performs the former), due to the way VPKs work. Assets may remain unchanged but move between VPK files during rebuilds of the game. The only way for XDelta3 to recognize that textures have moved, and to only internally relocate the data rather than replace it, is to tar it first so it treats it as one cohesive block until it is unpacked.

We account for user modifications to game folders by sending a list of expected “stock” files to the launcher over the web, relative to the game version that the user currently has. When preparing an update, developers could generate this list, with, e.g.:

```
ls -d1aA maps/* > maps.txt
```

Which produces a file that can be read by GNU Tar when using the `-T` flag. This allows to have a consistent and expected set of files in each archive.

In terms of platform support, it has been confirmed that, with any moderately recent version of GNU Tar, byte-for-byte hash-identical archives can be generated on both Linux and Windows Vista+. On Linux, the distro-supplied version can be used. On Windows, the MSys2-built version of GNU Tar can be trivially bundled and used by the launcher. To a large extent, this will work even between relatively old versions of GNU Tar, providing a lot of reliability, and the ability to use system-included versions of Tar on Linux.

See section 8.1 for a detailed description of how the update will likely work from a technical POV.

1.2 Fallback

Especially in its early stages of development, we cannot expect the delta process to always be robust enough to succeed across all the different TF2C installations. A fallback system will be necessary. Traditional patch archives, containing all changed files in their entireties, will be automatically generated with `diff -qr`, compressed with LZMA through Pixz. The process of downloading and applying the patch will be completely automated.

Pixz allows generating indexed archives that are a collection of smaller blocks, making parallel decompression possible, massively speeding up decompression time. The filetype internally is .tpxz, however as these utilities will be a part of the launcher, it will be unnecessary to require users to explicitly download any of these tools or otherwise be aware of the process.

2. GUI

Needs more thought. Probably using PyQt5 or wxPython. This isn't hard to implement, at least for basic intent, but there's a high ceiling regarding what we can expose through the GUI (at the most advanced level, exposing proxy and download throttling settings).

PyInstaller can include the GUI package in the launcher's executable without any problems (see section 6 for more information on distribution). PyQt5 and wxPython are both explicitly supported by it.

3. Downloading

Our diamond jewel in this respect is Aria2. Aria2 is a command-line download utility that, in conjunction with Metalink files, supports segmented downloading (i.e., downloading a file from multiple separate mirrors simultaneously), HTTP proxies for users who may require that, customizable download/upload speed throttling, integrated BitTorrent support, automatic file validation, automatic corruption repairing, and IPv6.

Consider: A user with an extremely fast (and unreliable) connection wants to download the game. As he starts downloading, it will first start accessing a list of pre-specified HTTP mirrors and requesting specific segments of the game's files. As it receives these segments, it will verify them in real-time as they're downloaded to ensure there's no issue.

It will not *solely* use HTTP mirrors though. Simultaneously, if a torrent is available for the same file, it will start connecting to BitTorrent seeders and peers, and download segments of the file through that route. All in all, it will make use of as many sources as possible to maximize download speeds for the user, and to naturally load-balance across all available mirrors.

This provides redundancy as well. If a mirror has failed, it will simply ignore it. This avoids having a single point of failure, increasing reliability and security substantially. The only real SPOF would be in the Metalink file itself, as the mirror hosting it could fail, or it could be compromised by an attacker that replaces it with a malicious file. You would ideally want three separate mirrors hosting the Metalink file, and the launcher would need to be able to access at least two of them, and verify that they're both serving the same file.

I'd advise any readers to look at the Metalink website and the Aria2 Github and README to understand the full extent of just how powerful, robust, and efficient this combined system is. This cements another aspect of Kachemak's design philosophy as well: Using external utilities wherever possible, rather than reinventing the wheel.

4. Launching

Need to consult with the TF2C developers to figure out how this is currently done and how it can be ported. There is obviously a meaningful difference between launching the game from the launcher and launching it from your Steam library, and it's important to figure out what's happening there.

5. Game integrity checking

Aria2 will automatically verify the downloaded archive and ensure that, at time of extraction, there is no corruption. If corruption is represent, it will redownload only the corrupted segments.

For later verification, we can leverage the existing functionality we use for delta updates, creating tars of various components of the game, and then checking the hashes of those resulting tars against what we expect. If we find corruption in one of them, we can probably just redownload an archive containing that component (such as texture VPKs) and extract it.

The alternative would be, for every game version, storing hashes of every single file, and hosting that files online so they can be redownloaded individually in case of corruption. I think this would be a lot more effort for only a modest benefit but it's possible if we want to substantially reduce the download size for users who run into corruption.

Possibly, and at the cost of keeping a permanent 3.6GB of extra data on the user's PC, we can just keep the full archive downloaded instead of removing it after extraction. If there's an issue with corruption, we can have an autorepair that simply checks this archive's integrity, and re-extracts it (overwriting existing files) if a user is having problems.

6. Launcher distribution and updates

6.1 Linux

By and large, we can use PyInstaller to pack the project into a single executable file that contains all relevant modules. Utilities such as XDelta3, GNU Tar, and Aria2 should come from the distribution instead of being included, both for security and simplification. The launcher should prompt the user to install these items if they aren't already installed. On major distros, we can automatically try to install them through Apt, DNF, Zypper, or Pacman (as applicable), otherwise if we do not recognize the user's distro then we should ask them to install these utilities manually.

6.2 Windows

As with Linux, we can use PyInstaller to build a static executable that contains everything except external utilities. We may want to look into signing the executable to get rid of the scary warning. We will need to distribute XDelta3, GNU Tar, and Aria2 with the launcher. GNU Tar is available on Windows through MSys2, requiring only a minuscule amount of additional dependencies to make it functional. I believe the others can be built directly for Windows.

A NullSoft installer can be included to unpack the launcher and create a shortcut for it on the user's desktop.

7. Installation

Ideally, with Kachemak, users will no longer download the entire game from the website, but will instead just download the launcher (probably through a guided NullSoft installation wizard that will unpack the launcher's files and create a desktop shortcut for it).

The launcher will autodetect the user's sourcemods folder and install into it without manual intervention. It will also need to have a button to automatically open the game's folder, so that a user will be able to easily find it afterwards for modding and other purposes. On Windows, this is fairly straightforward as a registry key exists that corresponds to a user's sourcemods folder which we can check and act on. On Linux, it should usually be ~/.steam/steam/steamapps/sourcemods/, but this is unreliable, particularly for users that have Steam installed through a Flatpak or Snap. We will need to investigate better ways to autodetect the sourcemods folder on Linux, or else just hardcode all possible known paths, and then probe to see which one is valid. An option for advanced users to manually choose the directory will be necessary.

Additionally, the process of semi-automatically creating a symlink to an external drive, on Windows or Linux, should be available for advanced users. This will only require that the user specifies where they want the folder to be on their external drive. Existing sourcemods should be moved to a temporary location before being moved to the new folder after the symlink is created, as to prevent accidental data loss.

8. Basic layout

When run with no arguments, it shows a help screen, printing the application version and a list of possible arguments, following typical GNU syntax for command-line arguments. No matter which argument is used, the launcher should always inform the user if an update is available, along with what flag to use to install it.

8.1 Actions

`--update / -u`

Checks if gameinfo.txt is present in the current directory. If not, it informs the user to move the launcher binary it to their TF2C directory. Later, it could be made location-agnostic if there's a reliable way to detect Steam install location.

Access a list of text files over the web that contain, in turn, lists of files to compress. These would exist for /maps/, /bin/, /media/, /vpks/ (split into three based on type), and /maps/. Run GNU Tar with all necessary arguments to make the resulting archives reproducible, and the `-T` flag to use the downloaded files.

All in all, this means seven tars will be generated, seven delta files will be downloaded, seven delta files will be applied through XDelta3, and seven updated tar files will be unpacked in place, with both the delta and the tar file removed after extraction is finished. If, at any stage, a problem occurs (most likely with a delta hash mismatch), we should give the user the option to install an update through the fallback mechanism instead. We should ensure that the user has at least 5GB of free hard drive space before running this operation, to ensure there's no risk of out-of-space errors causing unexpected problems, though it's very unlikely we'll come close to using a major portion of that 5GB during the upgrade.

`--install / -i`

Autodetect the user's sourcemods folder, if possible. If this is impossible, error out unless a separate flag has been set containing its path. Check the server for the most recent version of the game and download the corresponding archive through its Metalink via Aria2, and set it to download into a temporary directory. Maybe look into how Debian does this with their WATCH files as a way of checking for new releases and always getting the most recent one in a mostly automated manner. The temporary directory could be /tmp/ on Linux, but since it's often a RAM disk, we should find a way to make sure it will have enough space. If this is impossible, we can just run a check on available disk space (should already be possible through Aria2 directly?) and download to the hard drive (requiring enough for both the archive and the final extracted game simultaneously), deleting the archive after extraction is finished.

`--verify / -v`

Verify the sums of all game files, as elaborated in an above section.

`--version / -V`

Print launcher version. Maybe check and suggest if an update's available.

`--launcher-update / -l`

Update launcher to latest version, if one is available.

`--start / -s`

Start TF2C.

8.2 Modifiers

`--path / -p`

Specify path to your sourcemods folder.

`--gui / -g`

If/when implemented, launch Kachemak in GUI mode.

8.3 Future

1. Maybe options for manually specifying paths to dependencies like Tar and XDelta3? If bundled, this would let a user choose their system-wide versions instead. If not bundled, this would let a user manually specify the path if they want to use self-compiled versions, or if they literally aren't in the user's PATH by default.