MHLab

PATCH - Updating System

User manual

WELCOME

Welcome to the **PATCH - Updating System** documentation! Before diving into the software, I suggest taking a look at this manual.

CONTACTS

If you need help with the software or you just want to discuss something related to it, just contact me.

- Email: m4nu.91@gmail.com
- Discord: ManHunter#0637
- Discord Server: https://discord.gg/0ndGBjvogdY5Snlw
- Twitter: @MHLabSoftware
- Unity Forum: https://forum.unity.com/threads/p-a-t-c-h-ultimate-patching-system.342320/

WHAT IS IT?

PATCH - Updating System is a rock-solid, professional, all-in-one, smart and clean solution to manage and distribute updates for your games and applications.

Platforms

It is fully compatible with **.NET Core**, so it can run on all platforms .NET Core can run on. It comes with implementations for Unity3D, WPF and command line.

Main features

- patches are generated by a binary diffing algorithm
- strong patches compression
- corrupted/modified files repairing
- files attributes synchronization
- fully customizable UI

- no server-side code required: a normal HTTP server is enough
- sequential and non-sequential patches processing
- shortest path calculation for updates
- detection of very old versions with full-repair triggering
- self-update functionality

Admin Tools

It includes admin tools for your convenience:

- versions management
- patches management
- launcher updates management
- CI/CD friendly thanks to a command line tool

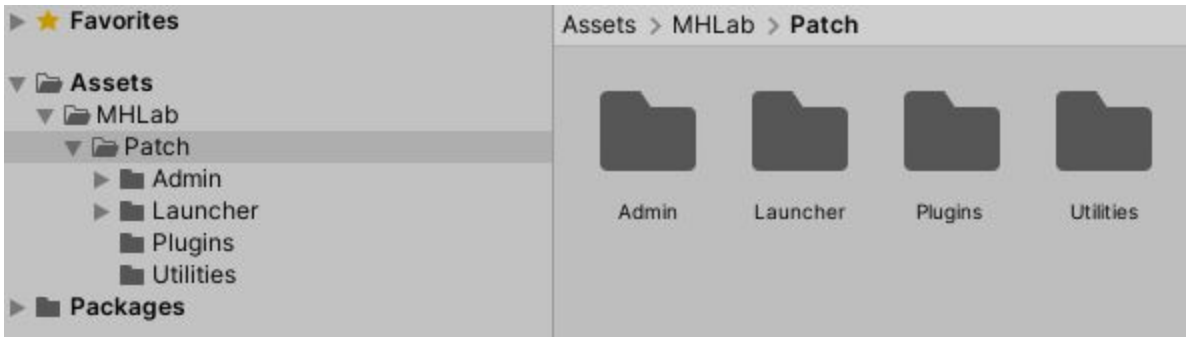
GETTING STARTED

The whole documentation assumes you are working on Windows, so commands and everything else refer to its environment. If you work on a different OS, don't worry: the process is the same, very minor changes in command line commands are needed.

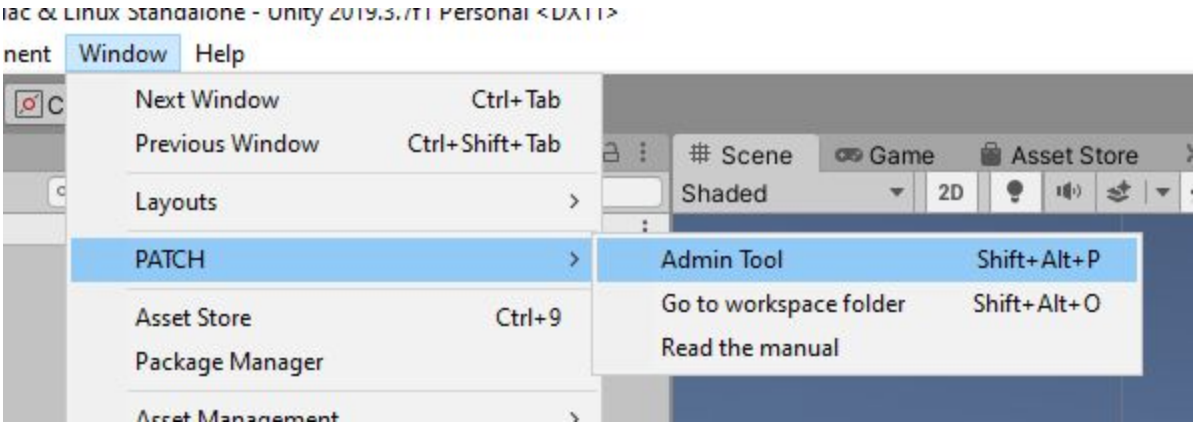
Also, it assumes you own the **PATCH - Updating System** package from Unity's Asset Store.

Installation

Just import the package in your project into Unity Editor. After importing, your project should contain at least these folders:



To check if it is correctly installed, go on the Window menu and make sure you can find the PATCH menu item.



How does this updater work?

Before we dive into the software itself, a little bit of theory. I know, I know: it is boring. But it is needed. In this way you completely understand what we are going to do here.

When you develop a game or an application, you also want the ability to deploy updates and fixes for it. There are multiple ways you can deliver updates for your software, but the most intuitive one in my opinion is the classic **Launcher**. The Launcher is a software that is **not embedded** into your own application/game: it is standalone and it is started **before** your game. It repairs corrupted/damaged files, it checks for updates, it applies updates, then it starts your game if everything is ready.

You may also need to deploy updates for the Launcher itself: so it also needs to **self-update**. In this package I propose two ways of self-updating: one for Unity and one for WPF.

Scenes

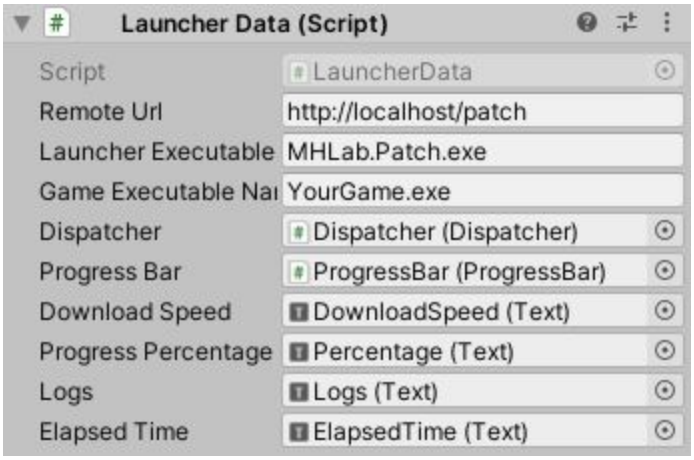
The package comes with an example of a classic updating architecture. You can find sample scenes in `MHLab/Patch/Launcher/Scenes`:

- **Launcher**. It is the scene that contains your Launcher logic and UI. It updates your game. In the hierarchy find a `LauncherData` game object: it contains settings for your `Launcher` script.
- **PreGame**. It is the scene that contains your Launcher self-updating logic and UI. It updates your Launcher. You need to set it as the first scene in your game. In the hierarchy find a `LauncherData` game object: it contains settings for your `LauncherUpdater` script.
- **SampleGame**. It represents your game, it is just a placeholder.

Setting up

To distribute updates you will need a web server or any other service that can serve files over HTTP. The Launcher's downloader can access files through direct URLs, like: http://yourIP/yourFolder/yourFile.exe, so make sure that your files host serves them **correctly**! For testing purposes you can also use a local web server like WAMP.

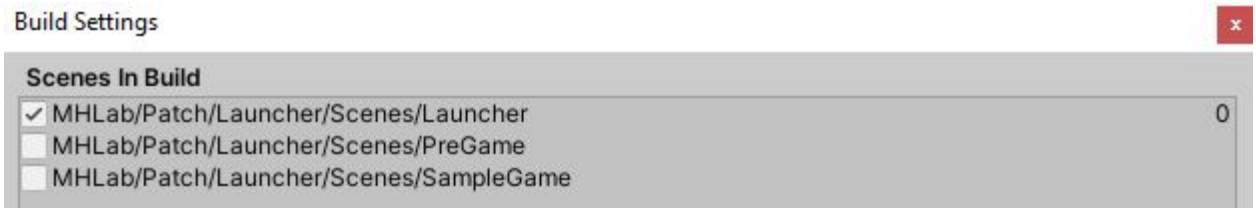
Decide what will be your remote workspace on your web server: I'll pick http://localhost/patch/ folder. So fill the LauncherData component with your settings in both Launcher and PreGame scenes.



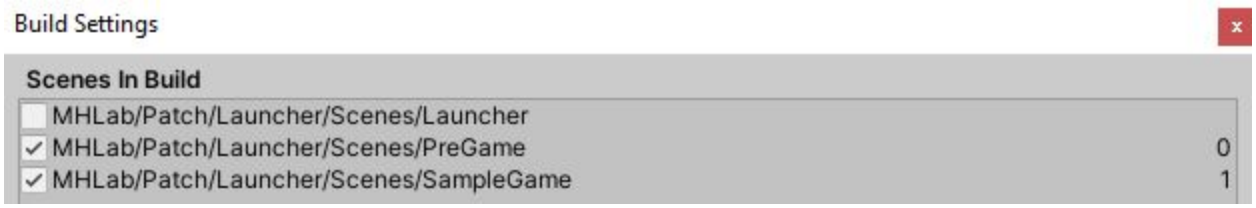
In the PreGame scene you will also find the LauncherUpdater game object: set the Scene To Load property inside its LauncherUpdater component. It is the index of your game scene in your build settings.

Build settings

When you build your Launcher, you have to **include** at least the Launcher scene at index 0.



When you build your Game, you have to **include** the PreGame scene at index 0. SampleGame represents your main game scene.

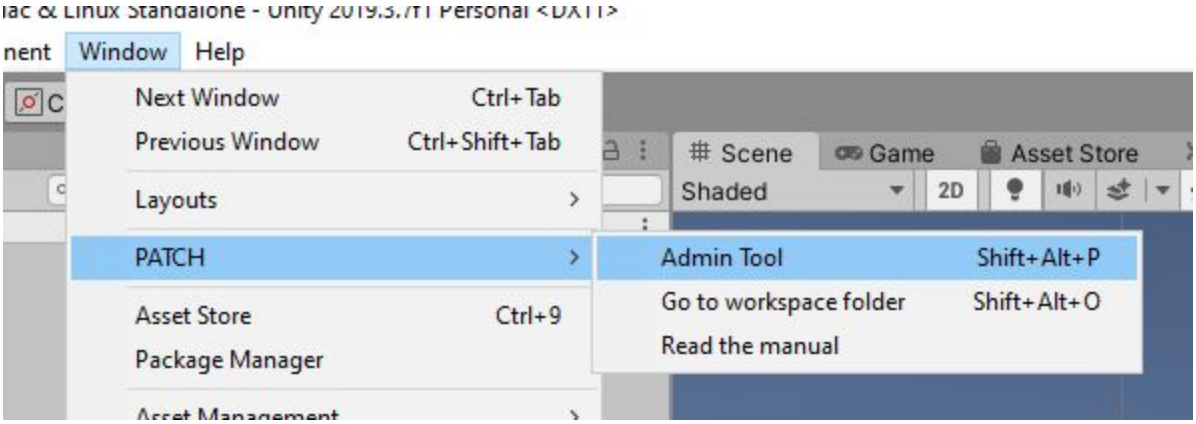


THE ADMIN TOOL

Before you can start updating your game, you have to perform some initialization tasks.

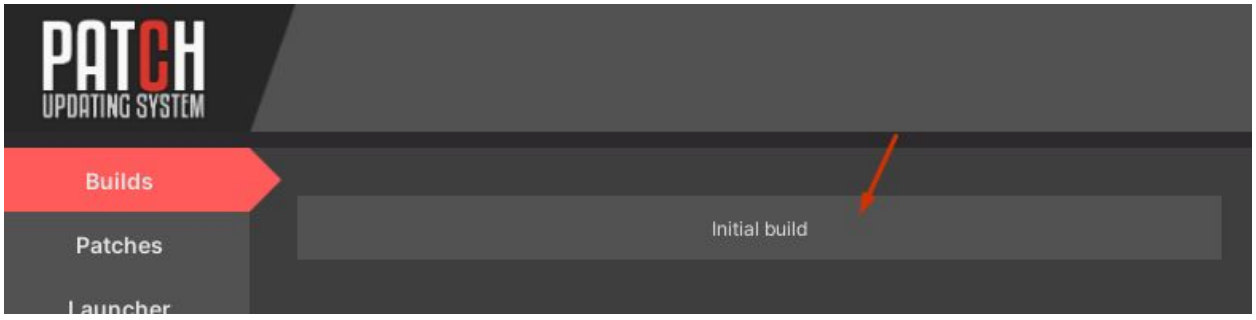
Initialization

Go on `Window > PATCH > Admin Tool` or press `Alt + Shift + P`: this will initialize the workspace. If you now click on `Window > PATCH > Go to Workspace` folder or press `Alt + Shift + O` the workspace will open and you will see some new folders: `App`, `Builds`, `Patches`, `Updater`.



The first game version

Now you just need to build your game as I explained here. Once you are done and built files are available, move all of them to the `App` folder (or build directly inside it) inside the PATCH's workspace. Go to `Admin Tool > Builds` and hit the `Initial build` button.



After the computation, you will notice new files and folder spawned in your Builds folder:

- 0.1.0 folder, it contains your game's files with some metadata about the version number
- builds_index.json file, it contains metadata about all versions
- build_0.1.0.json file, it contains metadata about the specific version

Upload the first version

It's a good time to upload files! Upload 0.1.0 folder, builds_index.json and build_0.1.0.json files on your web server.

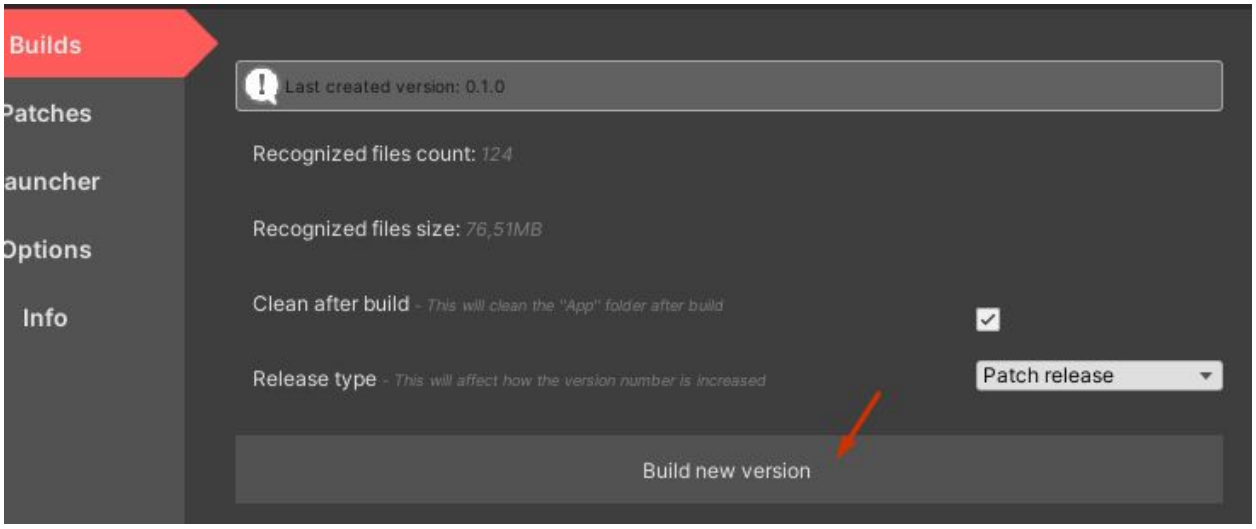
To test if your files are reachable by the Launcher, just navigate over them with a normal web browser. Assuming we uploaded files at http://localhost/patch/, we will navigate on http://localhost/patch/Builds/builds_index.json. The browser now should show a JSON string.

The second version

When you made enough changes to your game and you want to deploy a new version, just build your game again and put it in the App folder. Go to Admin Tool > Builds again. This time you can notice that more options exist. Before proceeding, you should take a look at Release type. It regulates how the version number is increased and contains three options:

- **Patch release**: with a previous version number of 2.3.6, a patch release will change the version number to 2.3.7. It is used for very little changes, hotfixes and similar stuff.
- **Minor release**: with a previous version number of 2.3.6, a minor release will change the version number to 2.4.0. It is used for minor changes or additions to the existing functionalities.
- **Major release**: with a previous version number of 2.3.6, a major release will change the version number to 3.0.0. It is used for major changes/additions to the existing functionalities or for changes/additions that break the backward compatibility.

Pick a release type according to your needs and hit the Build new version button. Again, after some computation the process will complete and you will find in your Builds folder a new version.



You can avoid uploading all versions you built. Patches are enough to update your clients' game. Why would you want to upload a build, then? Because of the **repair functionality**. Patches can be applied only if the game files' state is correct. So if your client corrupts a file or tries to modify it, the patch **cannot** be applied. The repairer runs exactly for this reason: if a file is corrupted, the correct state is **restored**. The repairer also runs if the client's version is really old. Really old = the local version does not exist on the server and the smallest available version is bigger than the local version.
Remember: when you want to upload a new build, you must upload always version-specific files and folders (`0.1.1` folder and `build_0.1.1.json`, for example), but also the index file (`builds_index.json`).

Generating the first patch

At this point you have two versions of your game. It's time to generate a patch between them: in this way updating from the old version to the new one will be a breeze. Go to `Admin Tool > Patches`. Set the version from / version to (it is automatically set on the last two versions, for your convenience) and the compression level, then hit the `Build new patch` button.

The compression level can go from 1 to 9. It impacts how strong the compression for that patch will be: lower levels offer better compression time but worse compression ratio.

When the process completes, you will find new files in your `Patches` folder (let's say your versions are 0.1.0 and 0.1.1):

- patches_index.json file: it tracks down all patches you built
- 0.1.0_0.1.1.json file: it contains metadata about the patch between version 0.1.0 and version 0.1.1
- 0.1.0_0.1.1.zip file: it is a compressed archive that contains actual files needed to apply this patch

Upload the first patch

Upload `0.1.0_0.1.1.zip`, `0.1.0_0.1.1.json` and `patches_index.json` files on your web server. Remember to always upload `patches_index.json` everytime you upload a patch.

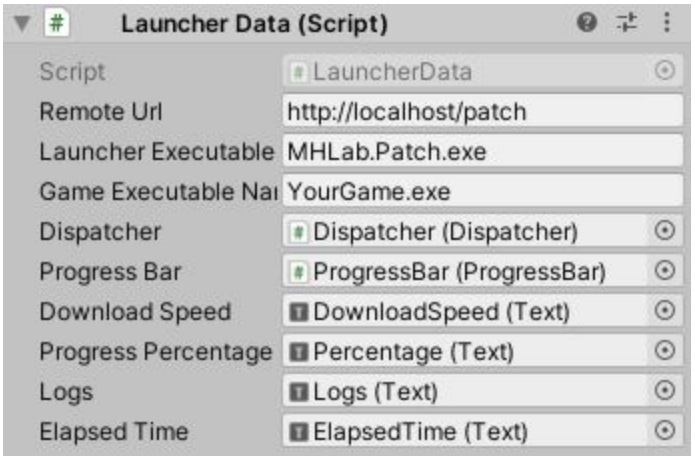
THE LAUNCHER

The Launcher is what your client will run to update the game. It requires an initial configuration in order to work.

Initialization

I will refer to the `Launcher` scene in `MHLab/Patch/Launcher/Scenes`, but the configuration is similar for other clients (WPF, WinForms, etc).

To configure it, just find the `LauncherData` game object: it contains a `LauncherData` component. It should look like this:



Here you must set some information:

- **Remote URL**: the HTTP address where you uploaded files previously
- **Launcher Executable Name**: the name of your Launcher executable, with the extension. In the screenshot you can see an example related to Windows's exe.
- **Game Executable Name**: the name of your game executable, with the extension.

Other settings you see in the screenshot are just for UI management.

If you need to deeper customize settings, take a look at the `Launcher.cs` script, in `CreateSettings` method. The `LauncherSettings` instance exposes multiple customizable settings you can tune to fit your needs.

Customization

Launcher and other scenes are just samples: you're encouraged to customize them. Also the Launcher.cs script is just a minimal sample of what you can do with the PATCH's API: feel free to edit it.

The UI can be trivially customized with the Unity UI system (or with Visual Studio if you're using the WPF version).

First launcher update

When you are satisfied with your Launcher, it's time to build it as I explained here. Place built files in the `Updater` folder in your PATCH's workspace then go to `Admin Tool > Launcher`.

Here just insert the `Launcher archive name` and the `Compression level` and hit `Build Launcher update`.

When the process completes you will notice a new file in the `Updater` folder: `updater_index.json`. It contains metadata for the Launcher update. PATCH also generated a compressed archive (it is in the root folder of PATCH workspace) that contains your Launcher, ready to be distributed to the clients.

Uploading the first launcher update

Just upload the whole `Updater` folder to your web server. That's it! :)

Test the launcher

If you want to test the Launcher, **just create a new folder** and extract inside it the content of the Launcher compressed archive. **Don't** run your Launcher inside the Updater folder!

Run the extracted executable and check that everything is acting like you expect!

After the first run, your Launcher downloaded your game in the `Game` subfolder. From now on you can run your game directly: the `PreGame` scene will check if any update is available and - if so - it will run the Launcher for updates.

YOU ARE READY TO GO

Congratulations! You completed the basic steps to use this software! This is just the beginning of your journey! If you want to read more about it or you need more advanced information, take a look at https://github.com/manhunterita/PATCH/wiki.

This online manual is updated more frequently and contains a lot of advanced information, so make sure to visit it!

Topics you will find:

- API explanations
- Customization advices
- Recommended softwares to support your development
- Best practices and strategies to handle your deploys

Thank you,

Emanuele - MHLab