P.A.T.C.H. - Ultimate Patching System

Professional Autopatcher Tool, Customizable and Hackable

An user guide to unleash the power of this software

Created by MHLab

PREFACE

Hello, I am Emanuele: the developer behind P.A.T.C.H.!

If you are here, probably you are interested to (or purchased) my solution for patches application and games updating.

My target is to offer the best patcher solution on the Asset Store (*yes, yes: I'm an optimistic guy :P*), an all-in-one solution that can offer all features you need out-of-the-box.
With your help I can improve it!

If you purchased it, I want to thank you! Your contribute will tempt me to improve a lot this software! So thank you for trusting in my solution, you will not be disappointed!

N.B: I will appreciate each review you will write for my product on Asset Store; each review can help other users to understand what this software can do for them! So please: write reviews!

If you didn't purchased it and you are here only to take a look at this solution, don't worry: this guide can be useful also for you! You will understand why this software is a must for your needs! :P

In both cases, I will guide you through the magic world of patches and games updates: I hope you will enjoy this fantastic journey!

Let's start!

INDEX

What is it?

If you are here, you probably know what this software is. But it's better to expose an official definition!

P.A.T.C.H. is a professional solution for applications/games patching and updating.

It contains all features you need: installer, patcher, launcher, repairer, patches creator, FTP uploader, etc. As I said: all-in-one, smart and clean solution! :)

It can generate very small patches thanks to its included file binary diffing algorithm.

What does it mean? It means that if you change only 5 bytes in your build, P.A.T.C.H. will create a patch that will change only those 5 bytes on users' builds, instead of downloading the entire edited file. Your users will be able to maintain updated their application copies with no pain or headaches, by saving bandwidth and time. Your users will love you (*platonically, of course :P*)!

In addition, you will be able to create small-sized patches (and upload them) quickly with included tools!

Features review

P.A.T.C.H. comes out with a lot of features, like:

- **Binary diffing** algorithm;
- Enables **very small** patches;
- Bandwidth **saving**;
- Secure checking for patches hash;
- **Strong** patches compression;
- Various type of compressions;
- Callbacks system to monitor what *P.A.T.C.H.* is doing;
- **Encrypted** configs;
- Linear and non-linear patches application;
- **Unity GUI** integrated;
- One-click-deploy;
- Upload builds and patches on **FTP**;
- **Installer** and **Repairer** feature;
- In-game **embedded** patcher
- Full **C#** coded;
- Customizable settings;
- Outstanding flexibility;
- Comes with launcher **source code**;
- If download fails, it will take care to **download again** failed patch for customizable amount of attempts;
- If patch process fails, it will take care to **rollback** your game to previous version;
- Launching **argument** to avoid obsolete clients;
- Files download over **HTTP, HTTPS**, **FTP** and **file system**;
- Integrated **localization** system;
- **Shortcuts** creation;

- Self-update feature
- Command Line Patch Builder
- **WPF** and **WinForms** examples too

How it works

How it is composed

P.A.T.C.H. is composed by more components that allow you to take advantage of it in your common updating tasks.

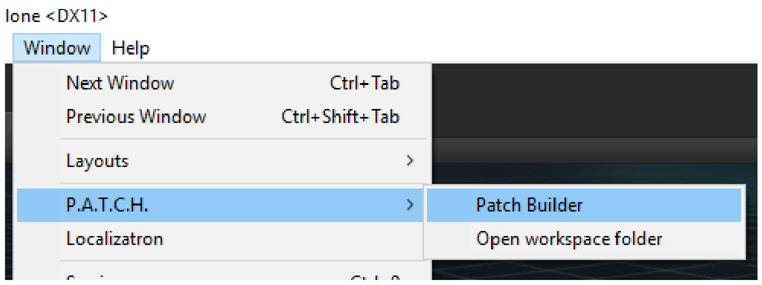
The main and major two components are Launcher and Patch Builder. They are also sub-divided in more components.

Launcher component will allow your clients to update, repair or install your game by downloading your updates.

Patch Builder will allow you to manage, create and upload patches, builds and deployed builds.

Patch Builder

Using the Patch Builder you can quickly and easily create your build versions, create a patch between two build versions, upload your files. You will find it in "*Window/P.A.T.C.H.*" top menu in Unity Editor:



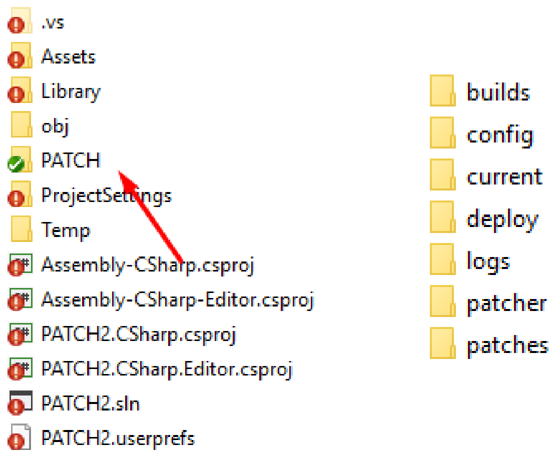
Tips & Tricks

In case you want to use the external version (made in .NET, with WinForms), you will find it in "*MHLab/PATCH/Source/PatchBuilder/PatchBuilderWinForms/PatchBuilderWinForms.zip*"! Extract it where you want on your file system (I recommend out of Unity project!). The workflow is the same!

Tips & Tricks

You can click on "*Open workspace folder*" to have a quick access to P.A.T.C.H.'s folder structure!

When you open it for the first time, a folder structure will be created for your game project.
This folder structure will be created in your Unity project root, under "*PATCH*" directory.



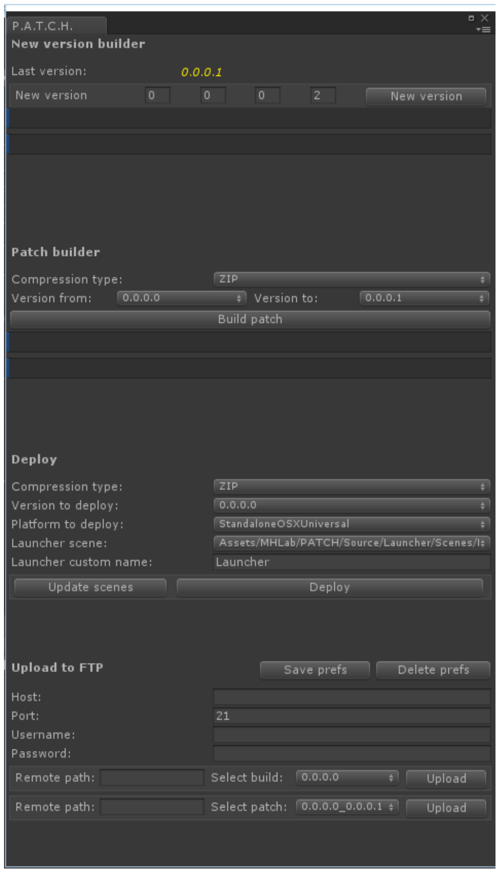
Each directory has its own purpose, so let's explain their meaning! By learning their purposes, you will able to easily move between them!

PATCH: it is the P.A.T.C.H.'s workspace for your current Unity project, in it you will find all directories used by P.A.T.C.H. You can access it by clicking on "Open workspace folder".
builds: this folder contains your built versions and index files
config: this folder contains configuration files, they are used only for WPF/WinForms version
current: in this folder you will put your current build to compute with Patch Builder
deploy: this folder will contain your deployed packages for versions you selected
patcher: in this folder you have to put you built Launcher (Unity one or WPF/WinForms one: no matters, it depends on what you chosen to use) if you want to allow self-update feature. Check Self-update section for more informations
patches: this folder contains all patches you will create, relative index files and a versions.txt file
logs: if an error occurs, in this folder you will find a log file

Tips & Tricks

If you use the external version of Patch Builder, you will notice also a "*config*" directory. This folder will contain a *config.xml* file (you can generate it with "Create config" button) and its encrypted copy.

After this, this window will appear:



P.A.T.C.H.

New version builder

Last version: *0.0.0.1*

New version 0 0 0 2 New version

Patch builder

Compression type: ZIP

Version from: 0.0.0.0 Version to: 0.0.0.1

Build patch

Deploy

Compression type: ZIP

Version to deploy: 0.0.0.0

Platform to deploy: StandaloneOSXUniversal

Launcher scene: Assets/MHLab/PATCH/Source/Launcher/Scenes/l

Launcher custom name: Launcher

Update scenes Deploy

Upload to FTP Save prefs Delete prefs

Host:

Port: 21

Username:

Password:

Remote path: Select build: 0.0.0.0 Upload

Remote path: Select patch: 0.0.0.0_0.0.0.1 Upload

This window contains all tools you need to manage your game updates! Let's see how they work!

NEW VERSION BUILDER

This tool will allow you to create a P.A.T.C.H.-ready version of your built game.

It is very easy to use:

- build/put your game into "***current***" directory
- choose your version for your current build
- click on "***New version***" button

After a little computation time, your output will be located in "***builds***" directory. It will be composed by:

- your built game
- an encrypted "***version***" file

This file is extremely important: it contains your game current version, the patcher will always use it!

There are also other files in your "builds" directory:

- index
- index_YourVersion.bix

These files will be useful when we want to activate Installer or Repairer feature.

PATCH BUILDER

When you have two or more versions of your game, *Patch builder tool* will be available.

This tool can create a patch (a binary diff file) between two selected versions of your game.

To make it works, you only need to:

- select a compression type

- select version from and version to
- click on “***Build patch***” button

When computation is done, you will find a new patch in your “***patches***” directory. A patch is composed by:

- an **archive** file (yourVersionFrom_yourVersionTo.archive)
- an **index** file (yourVersionFrom_yourVersionTo.pix)
- an entry in **versions.txt**

The archive contains all difference data between your selected versions, the index contains information about files and versions.txt contains a list of available patches.

Deploy

Now you're probably asking yourself: “*ok, now I have all stuff done, how can I include a patcher in my build?*”

Deploy tool (or One-click-deploy) will allow you to create an archive with your selected version of your game and the patcher scene.

So if you built **atleast** a version of your game, you will be able to see this tool enabled.

To work with it, you only need to:

- select a compression type for your output archive
- select the version of your game you want to deploy
- select a platform for your patcher building process
- select your launcher scene
- type the name of your launcher
- press “***Deploy***” button

After computation, you will find a compressed archive in your “***deploy***” folder.

This archive is users-ready (you can send it to your users!) and it contains:

- the selected version of your game (your game build + version file)
- your built launcher

Tips & Tricks

You can also perform this task manually, if you prefer. Just create your Launcher build and zip it together with a version of your game (you can find versions in “builds” directory).

TIPS & TRICKS

These settings for Launcher deployment will be used also to upload your Launcher to remote host, in case you need.

UPLOAD TO FTP

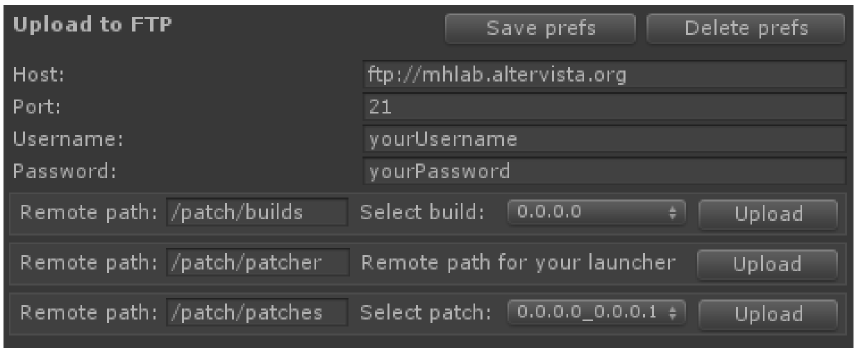
At this time, you're ready to go. But what is missing yet?

The smartest among you will say: “*we didn't uploaded nothing yet on our remote hosting*”.

You're right! So we can analyze the last tool!

The uploader tool is simple and intuitive, fill all fields with your FTP credentials and save them with “***Save prefs***” button: next time you will open P.A.T.C.H. your credentials will be re-loaded automatically.

This is an example of how to configure credentials:


Upload to FTP

Host: ftp://mhlab.altervista.org
Port: 21
Username: yourUsername
Password: yourPassword
Remote path: /patch/builds Select build: 0.0.0.0
Remote path: /patch/patcher Remote path for your launcher
Remote path: /patch/patches Select patch: 0.0.0.0_0.0.0.1

When you complete the configuration, just:

- select the build/patch you want to upload
- press “Upload” button

For Launcher you only need to set the remote path, because all information will be extrapolated from Deploy section!

<u>Take in mind these remote paths, you will need them in Launcher configuration!</u>

Of course you can upload builds, Launcher and patches manually, if you are comfortable with.

To upload a build manually (in example, 2.0.3.1 version):

- create the remote path for builds (so create all needed folders)

- upload index file and index_2.0.3.1.bix to created path
- upload the whole 2.0.3.1 folder to created path

To upload a patch manually (in example, 2.0.3.1 to 2.0.3.2):

- create the remote path for patches (so create all needed folders)
- upload 2.0.3.1_2.0.3.2.archive and and 2.0.3.1_2.0.3.2.pix to created path
- upload versions.txt to created path

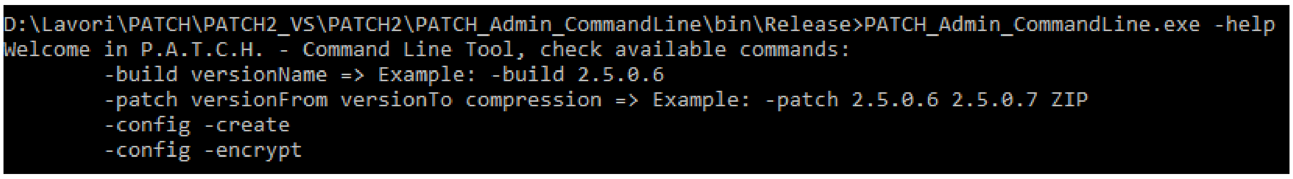
Command Line Patch Builder

A command line version of Patch Builder is available for users who needs a command line interface to build automatically their patches/builds in a Continuous Integration environment.

You can find Command Line Patch Builder in "*MHLab/PATCH/Source/PatchBuilder/CommandLinePatchBuilder/CommandLinePatchBuilder.zip*".

It has exactly the same folder structure I explained before in this document.

You can run it from your favourite command line by writing:



```
D:\Lavori\PATCH\PATCH2_VS\PATCH2\PATCH_Admin_CommandLine\bin\Release>PATCH_Admin_CommandLine.exe -help
Welcome in P.A.T.C.H. - Command Line Tool, check available commands:
        -build versionName => Example: -build 2.5.0.6
        -patch versionFrom versionTo compression => Example: -patch 2.5.0.6 2.5.0.7 ZIP
        -config -create
        -config -encrypt
```

With "-help" parameter you can see a list of all available commands. So, if I want to create a build with the content of "current" folder, I'll just write:

PATCH_Admin_CommandLine.exe -build 2.0.1.5

Instead of *versionName* you can write "***auto***" to auto-complete the command with latest version.

You can create a patch between two builds with:

PATCH_Admin_CommandLine.exe -patch 2.0.1.5 2.0.1.6 ZIP

Instead of two version, you can write "***previous***" and "***current***", to refer respectively to previous build and current build.

LAUNCHER

Launcher is the client part of P.A.T.C.H.! With this tool you can:

- update your users' games in an easy and comfortable way
- install your game the first time you open it
- repair a corrupted game
- launch your game with an argument, to avoid obsolete clients

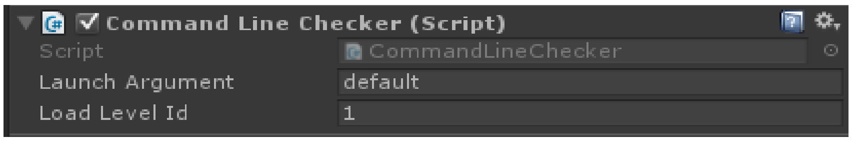
You can find some Launcher example scenes in *"Assets/MHLab/PATCH/Source/Launcher"* folder.

TIPS & TRICKS

If you want to use .NET Launcher version, you will find an example Launcher and its source in the same directory.

There are four examples here:

- LaunchArgumentChecking, this scene demonstrates how to make an argument checking on your game. You can put it as first scene to validate your game: close it if runned alone, continue to run it if runned by Launcher. You can customize settings by editing CommandLineChecker



Command Line Checker (Script)

Script	CommandLineChecker
Launch Argument	default
Load Level Id	1

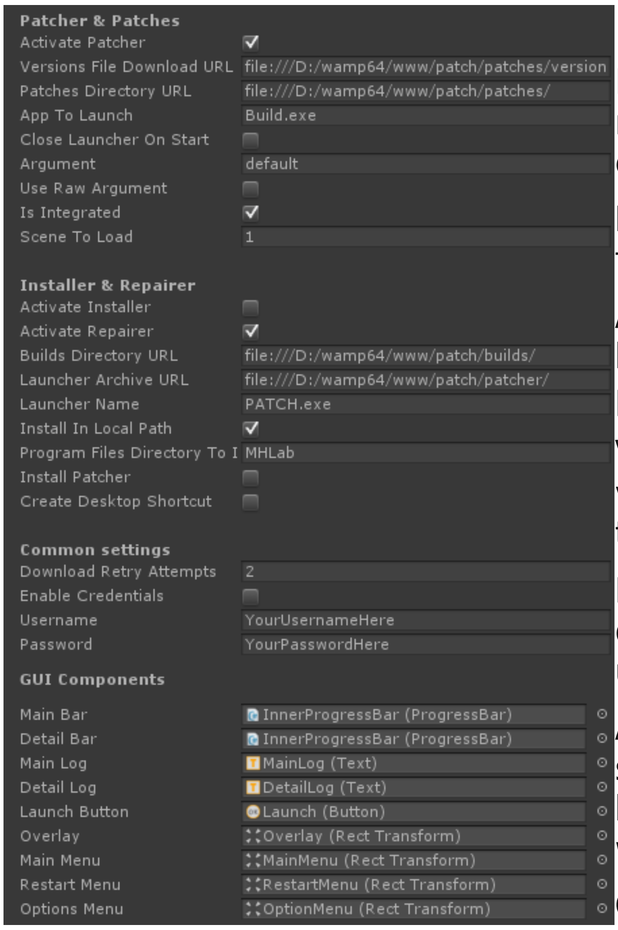
  object:
- LauncherStyle1 and LauncherStyle2, these scenes contain example Launcher skins
- LauncherWinFormsStyle, this folder contains sources and a demo of a .NET Launcher

Let's see how to set a Launcher to work with your games!

Launcher configuration

Choose your favorite scene between LauncherStyle1 and LauncherStyle2, we will use one of them for testing purposes, but feel free to customize them as you wish.

As you can see, there is a gameobject called “*Launcher*” with a script Launcher.cs attached:



Patcher & Patches

Activate Patcher	✓
Versions File Download URL	file:///D:/wamp64/www/patch/patches/version
Patches Directory URL	file:///D:/wamp64/www/patch/patches/
App To Launch	Build.exe
Close Launcher On Start	☐
Argument	default
Use Raw Argument	☐
Is Integrated	✓
Scene To Load	1

Installer & Repairer

Activate Installer	☐
Activate Repairer	✓
Builds Directory URL	file:///D:/wamp64/www/patch/builds/
Launcher Archive URL	file:///D:/wamp64/www/patch/patcher/
Launcher Name	PATCH.exe
Install In Local Path	✓
Program Files Directory To I	MHLab
Install Patcher	☐
Create Desktop Shortcut	☐

Common settings

Download Retry Attempts	2
Enable Credentials	☐
Username	YourUsernameHere
Password	YourPasswordHere

GUI Components

Main Bar	InnerProgressBar (ProgressBar)
Detail Bar	InnerProgressBar (ProgressBar)
Main Log	MainLog (Text)
Detail Log	DetailLog (Text)
Launch Button	Launch (Button)
Overlay	Overlay (Rect Transform)
Main Menu	MainMenu (Rect Transform)
Restart Menu	RestartMenu (Rect Transform)
Options Menu	OptionMenu (Rect Transform)

Each field has a tooltip to explain its meaning: you can see it by staying with cursor on desired field.

I want to explain their meanings anyway, for your comfort!

Activate Patcher: If enabled your launcher will provide to check and apply patches to your current build.

Versions File Download URL: Your versions.txt file remote URL. Usually it is the same for patches.

Patches Directory URL: Your patches directory remote URL. The same you uploaded your patches.

App To Launch: Your game name! This string will be attached to app root path to launch your game when patching process will end!

Close Launcher On Start: Determines if your launcher will be closed after your game starts!

Argument: This argument will be attached to your game running command! It should be the same of your CommandLineChecker setting, don't forget it!

Use Raw Argument: If enabled your argument will be sent as raw text, if not your argument will be sent as "YourGame.exe –LaunchArgs=YourArgument".

Is Integrated: If enabled, this will inform P.A.T.C.H.'s core that your Launcher is embedded in your game. Refer to “Embedding your Launcher in your game” section.

Scene To Load: If IsIntegrated is enabled, this will specify what scene will be loaded when update process is completed

Activate Installer: If enabled your launcher will try to install your build files before patches checking.

Activate Repairer: If enabled your launcher will start to check files integrity before patches checking. It is useful to fix files corruption of your users' builds!

Builds Directory URL: Your builds directory remote URL. The same you uploaded your builds.

Launcher Archive URL: Your launcher archive remote URL.

Launcher Name: Your launcher name!

Install In Local Path: If enabled your installer will install locally your game, if not your installer will install your game in *Program Files/ProgramFilesDirectoryToInstall* directory.

Program Files Directory to Install: If your installer have to install your game under Program Files folder, this will be the name of your game directory!

Install Patcher: If enabled your installer will install also a patcher. It is useful for standalone installers. Use it combined with "*Install in local path = false*".

Create Shortcut: If enabled your installer will create a shortcut to your patcher on desktop.

Download Retry Attempts: How many times downloader can retry to download a file, if an error occurs?

Enable Credentials: Enables WebRequests or FTPRequests with credentials. Generally, you need this when your remote directories are protected by login or your remote URLs are FTP ones!

Username: Username for your requests.

Password: Password for your requests.

GUI Components are already setted in example scenes, you can check them!

You need to set up all these fields to fit properly with your needs!

IMPORTANT! Remember to switch your API Compatibility Level from "***.NET 2.0 Subset***" to "***.NET 2.0***", before to build the Launcher! You can find this setting in *Edit > Project Settings > Player > Other Settings*.

After Launcher configuration, you can proceed with Deploy process explained before in this document!

At this point, you are able to test your Launcher and your patches application process.

You can unzip your deployed archive in a test directory and you can run the Launcher!

TIPS & TRICKS

The Launcher performs a lot of operations on files. So a good practice is to run it as administrator: if not, some users with limitated privileges can notice some issues.

TIPS & TRICKS

Sometimes, antivirus softwares can block the execution of your Launcher (because it performs operations over files). Just add your Launcher to antivirus' exceptions!

Launcher's components

Launcher is composed by multiple components to accomplish to various tasks: let's explain them in detail!

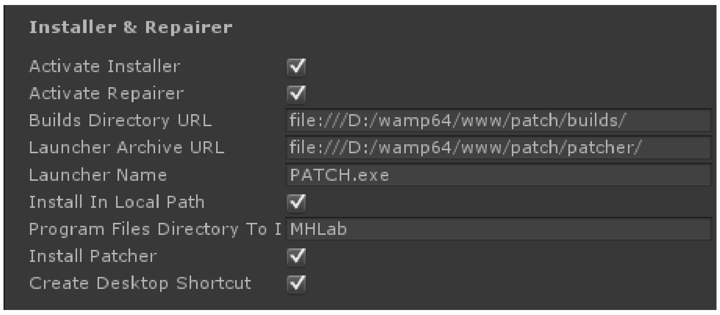
Installer

Installer feature will allow your Launcher to be deployed with no builds attached. Your Launcher will be able to download the **latest** version available on your remote host and then **apply** all patches available for that version.

APIs are simple, you can check an example of their usage in *Launcher.cs* script.

With this feature you can also create a standalone installer that will only install your version and the latest Launcher available.

The only limit is your mind! :P



Installer & Repairer

Activate Installer	✓
Activate Repairer	✓
Builds Directory URL	file:///D:/wamp64/www/patch/builds/
Launcher Archive URL	file:///D:/wamp64/www/patch/patcher/
Launcher Name	PATCH.exe
Install In Local Path	✓
Program Files Directory To I	MHLab
Install Patcher	✓
Create Desktop Shortcut	✓

As you can see, there are various settings in *Installer section*. Let's write something about them.

As first thing, your installer is triggered when the software can't find a valid ***version*** file. When this happens, it will check for the latest remote version available (in your *Builds Directory URL*), so it will proceed with Repairer feature.

If you don't want this component, just uncheck ***Activate Installer***.

Install in Local Path will allow you to install your game in current Launcher's folder or, if unchecked, in "*Program Files(x86)/MHLab*" (as example the previous screenshot, edit "*Program Files Directory To Install*" to change your directory).

Install Patcher will download your latest remote Launcher, it is useful if you want to create an Installer instead of a Launcher.

Create Desktop Shortcut can create a shortcut to your Launcher (or your game if Launcher cannot be found): *this feature is for now randomly bugged on Unity, because Mono misses some method implementations. Use it only in .NET Launcher! I will try a workaround in next release, check the Roadmap!*

Repairer

Repairer feature is triggered when the software can find a valid *version* file. In this case, the Launcher will check your local files and download all missing files or replace corrupted ones.

This can **avoid** all type of local users' interventions over your game files, because it runs every time you open the Launcher. If you combine it with *CommandLineChecker* script, your users will be not able to run a load-time-hacked game.

If you want to deactivate this component, just uncheck ***Activate Repairer***.

Patcher

Patcher feature is triggered after Installer/Repairer features. When a version is correctly installed (or a version is correctly validated), your Launcher will be able to check for latest patches available for your new installed version.

Patcher also has a rollback feature, to restore the latest original version if an error occurs (with no additional downloads, the whole rollback process is done locally!).

WPF and WinForms Launchers

As you certainly noticed, there is also a WPF/WinForms version of P.A.T.C.H.'s Launcher. The workflow is the same, but to use it you need to create an encrypted config file.

It is really simple:

- generate a fresh config.xml file with WinForms Patch Builder (*MHLab/PATCH/Source/PatchBuilder/PatchBuilderWinForms*), by clicking on "*Create config file*" button: it will be generated in your *config* directory, under P.A.T.C.H. workspace
- open config.xml file and fit it with your configuration
- click on "*Encrypt config file*" in your WinForms Patch Builder
- copy your encrypted *config* file in your WPF/WinForms Launcher folder (the same folder of .exe): you will find your *config* file in your *config/Encrypted* folder, under P.A.T.C.H. workspace

You can easily customize your .NET Launcher with included source codes. They are stored in *MHLab/PATCH/Source/Launcher/Scenes/LauncherWinFormsStyle* and *MHLab/PATCH/Source/Launcher/Scenes/LauncherWPFStyle*.

That's it! Enjoy also your .NET Launcher!

Self-Update feature

If you want, you can allow your Launcher to patch itself and its DLLs. This is extremely useful if you're editing your Launcher's skin or if you want to apply changes to your users' Launchers with no efforts (as example, if a new P.A.T.C.H. release comes out, you can simply make a patch for your users).

You can do this because from its v2.1.0, P.A.T.C.H. is able to patch executing/locked files.

When this happens, internal Launcher's state is marked as "dirty" and a Launcher restart is recommended.

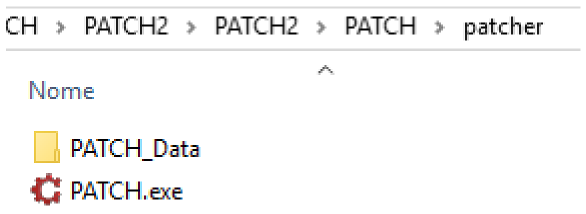
You can see this in *MHLab/PATCH/Source/Launcher/Scenes/IngameIntegratedPatcher* example: it will ask you for a restart and it restarts automatically the Launcher if you agree.

What do you need to do to enable self-updating?

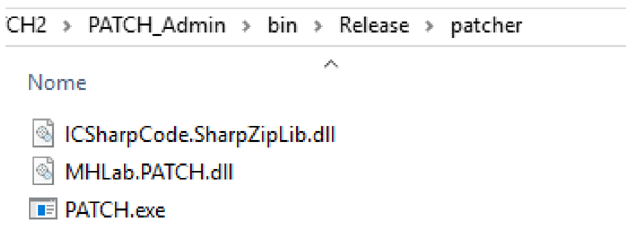
As you can see, in your P.A.T.C.H.'s workspace you can find a "*patcher*" directory.

Put in this folder your built Launcher before you create a new game version, so you can replace it each time you want to make changes.

IMPORTANT: when you put your Launcher in "*patcher*" directory and so you make a game version, you are not allowed anymore to delete it from "*patcher*" directory, because if you create a game version with no Launcher in it, your Launcher will delete itself after patch process.



An example of "*patcher*" directory for Unity version Launcher


CH2 › PATCH_Admin › bin › Release › patcher
Nome
ICSharpCode.SharpZipLib.dll
MHLab.PATCH.dll
PATCH.exe

An example of "*patcher*" directory for WPF/WinForms Launcher

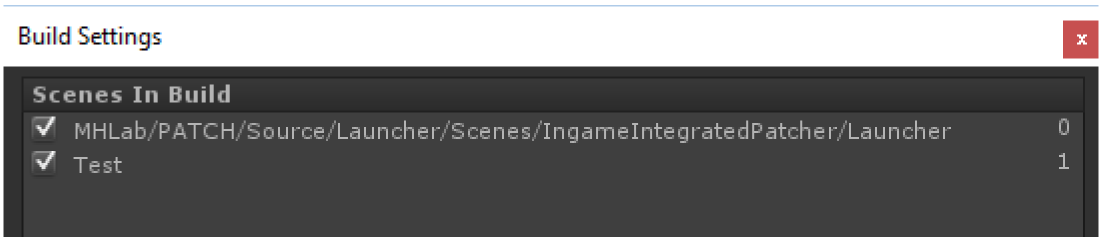
This feature is pretty useful if you're using external Launcher, not embedded in your game.

Let's see how to embed your Launcher in your game.

Embedding the Patcher in your game

If you want to embed the Launcher in your game you can forget about self-updating steps: they will be executed automatically.

It is pretty simple: create your Launcher scene in Unity and build it with your game, as first scene loaded.



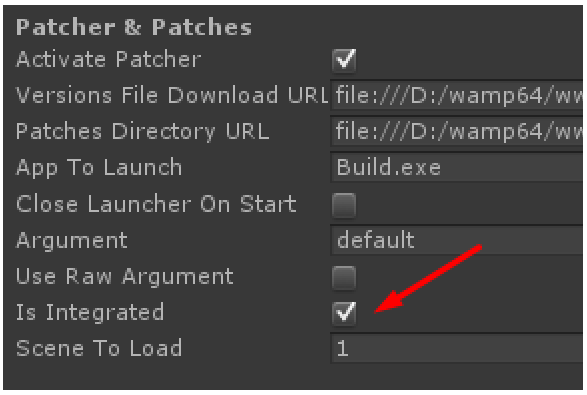
Build Settings

Scenes In Build

✓	MHLab/PATCH/Source/Launcher/Scenes/IngameIntegratedPatcher/Launcher	0
✓	Test	1

You can find an example scene here: *MHLab/PATCH/Source/Launcher/Scenes/IngameIntegratedPatcher*.

The only one thing you need to set is "IsIntegrated" flag: check it to allow your Launcher to be embedded in your game. If checked, you should also indicate which scene Launcher has to load after patch process!



Patcher & Patches

Activate Patcher	✓
Versions File Download URL	file:///D:/wamp64/ww
Patches Directory URL	file:///D:/wamp64/ww
App To Launch	Build.exe
Close Launcher On Start	☐
Argument	default
Use Raw Argument	☐
Is Integrated	✓
Scene To Load	1

When done, you can proceed by building your project in "*current*" folder, as previously explained, and make patches as usually.

When you run one of your built project, it will patch itself and embedded Launcher!

That's it, nothing else! Enjoy! :)

F.A.Q.

When I create a build and put it in "current" directory, I need to include also launcher or version/config files?

No, you don't need to include those files in your "current" directory. P.A.T.C.H. will automatically provide to include them when it is necessary. See relative chapter.

Can I use your patching system for projects unrelated to Unity?

Yes, you can. *P.A.T.C.H.* is built to work well with any application. If your system can run your Launcher, you can use *P.A.T.C.H. to update your application*.

Can I develop my own launcher logic and skin?

Yes, you can. Just open launcher project or Unity scene and edit it.

I developed a skin for your P.A.T.C.H.! Can I sell it on Assets Store?

Yes, you can. Develop what you want and sell it! It will help my product to gain a good visibility and help you to earn much money! This is a reasonable exchange!

Of course **you can't include my core files** in your package, but I think this is obvious... :P

Can I use it with Linux or Mac?

Yes, you can. With Unity integrated version you can compile your build for Windows, Linux or Mac by switching platform in your Unity Build Settings.

With standalone version it is a bit different: at this time, standalone *P.A.T.C.H.* comes with a precompiled Launcher that runs only on Windows. This doesn't mean that you can't run it on other platforms. :)

Attached to package there is full Launcher source code, you can open it with Xamarin/Mono and compile it targeting your favourite platform! This will make the trick!

What about licensing?

On *P.A.T.C.H.*'s Unity Asset Store page you can see the quote "This extension requires one license per seat". I know this is overkilled for indie studios and can be workarounded by making patches on a single seat, so I decided to distribute it with another licensing type.

You need just one license per deployed project. If you have 3 published games that are using *P.A.T.C.H.*, you need 3 licenses.

Does P.A.T.C.H. work also on mobile platforms like Android or iOS?

No, at this time *P.A.T.C.H.* can't officially manage mobile platforms due to different app management by those OSs. Look at roadmap to see future improvements planned. Of course, you can modify it to make it works on your favourite OS!

Do I need any particular settings in my Unity Editor to make P.A.T.C.H. works?

The only thing you need to set is the API Compatibility Level. By default, Unity Projects has this value setted to "***.NET 2.0 Subset***". You need to switch it from its default value to "***.NET 2.0***". You can find it in *Edit > Project Settings > Player > Other Settings*.

Do I need particular privileges on my OS to make P.A.T.C.H. works?

It is a good practice to run Unity as **administrator** to work with Unity integrated Patch Builder, in case your projects are under OS's special folders.

Regarding Launcher, due to files-editor nature of this software, you should always run it as administrator **to avoid** issues with your user's privileges.

How can I introduce localization in my Launcher?

P.A.T.C.H. includes a self-made localization system with its own documentation. Please refer to it to discover how to localize your Launcher. Look at: "*MHLab/PATCH/Localizatron/Doc*".

Customers support

P.A.T.C.H. is already tested, but bugs can always find a way to annoy you. If you can't solve an issue by yourself (be sure you read this document **one more time**!), please consider to report your bug to me: I will be happy to **improve** my software and **fix** bugs!

In your bug report, please **include** all information you think I need. In example: your Unity version, your OS, your installed plugins, project configuration, special folders used, etc.

You can also include your project environment (if possible, of course), so I can test the issue with your configuration.

Feel free to contact me (and nag me to death ;]), I will solve your problems as soon as I can (some delays can happens, based on different timezones)!

Email: m4nu.91@gmail.com

Skype: manhunterita

Twitter: @MHLabSoftware

Discord: https://discord.gg/0ndGBjvogdY5SnIw

Unity Forum: http://forum.unity3d.com/threads/p-a-t-c-h-ultimate-patching-system.342320

I also offer a **TeamViewer** or **Supremo** support, in case you can't make P.A.T.C.H. works properly by yourself! Just ask for it!

If you have suggestions or advices (on this document or on the software) feel free to message me with no fear!

Thank you again,

Emanuele - MHLab

ROADMAP

P.A.T.C.H. is constantly under improvement, so I write what I planned to introduce in this solution in near future. Next features to add:

- ~~Test and fix on Android and iOS~~ (failed)
- ~~Test and fix on Mac~~ (done)
- ~~Improve current download manager~~ (introduced in v2.0.3)
  - ~~Add download progress~~ (introduced in v2.0.3)
  - ~~Add download speed~~ (introduced in v2.0.3)
  - Add download restore after patcher closes
  - Add download pause
- File blacklisting or something like .gitignore
- ~~Create self-update feature~~ (introduced in v2.1.0)
- ~~Create documentation for installer and create examples for it~~ (introduced in v2.0.5)
- ~~Create documentation for installation repairer and create examples for it~~ (introduced in v2.0.5)
- ~~Create in-game integrated patch process~~ (introduced in v2.1.0)
- ~~Create automatic patches/builds FTP uploader~~ (introduced in v2.0.4)
- Create one-click-deploy for standalone version with standalone patcher
- Add additional compression methods for patches and builds (LZMA2, LZ4)
- ~~Add command line interface~~ (introduced in v2.1.2)
- On-the-fly patch for unused assets
- ~~Introduction of patch notes~~ (introduced in v2.1.0)
- Create a multiplatform builds directories structure (so Win builds, Linux builds, etc)
- ~~Add button in Unity Menu for quick opening of PATCH folder~~ (introduced in v2.0.5)
- ~~Add option to close patcher after game start~~ (introduced in v2.0.5)
- Instead to build entire new builds, introduce small hotfixes that don't require an entire new version to be made? Like the ability to only patch specific files?

- Introduce Mega.co.nz's API to download/upload patches and game versions
- Check for readonly files attribute and remove it
- Installer: prompt users to ask where to install the game/launcher
- Create something like Battle.NET: an app hub that can manage multiple apps
- Create an integrated anticheat solution

CHANGELOGS

v2.1.3:

- CORE: check for remote service available when Launcher starts
- API: added public *MHLab.PATCH.Utilities.Utility.IsRemoteServiceAvailable()* method

v2.1.2:

- CORE: some compatibility fixes for Unity 5.4 in code base
- CORE: fixed versions sorting in Patch Builder
- FEATURE: added command line tool

v2.1.1:

- CORE - IMPORTANT: fixed a directory issue on New Version creation when a Launcher is found in "*patcher*" folder
- CORE: fixed Create Desktop Shortcut flag

v2.1.0:

- CORE - IMPORTANT: Solved platform dependent issue for root path on Linux
- CORE - IMPORTANT: Introduced in-game embedded patching process
- CORE - IMPORTANT: Introduced self-update feature
- CORE - IMPORTANT: Introduced locked/running files update
- CORE: Introduced "*IsDirty*" internal state to communicate when Launcher needs to be restarted
- CORE: Introduced new WPF Standalone Launcher
- CORE: Introduced news system
- CORE: Introduced patch notes
- GUI: Introduced options and tools for Launcher
- CORE: Added "*Force repairing*" feature
- CORE - IMPORTANT: Solved privilege issue for OSX after patch process
- CORE: now Patch Builder creates an empty *versions.txt* when opened for the first time

v2.0.5p1:

- CORE: Solved platform dependent issue for root path on Mac

- CORE: an empty “*versions.txt*” now will result in a “*No patches available*” instead of a fatal error

v2.0.5:

- CORE: Solved pending warnings in FULL version
- CORE: Added MHLab.PATCH.Settings namespace to Localizatron settings files, to avoid naming collisions
- GUI: Added button in Unity Menu to quickly open workspace folder
- GUI: Added flag to close Launcher on game start
- GUI: Added multiple flags to activate/deactivate features
- GUI: Adjustments to Launcher script inspector
  - All fields are divided by categories
  - Added tooltips to help users to better understand what each field can do
- CORE - IMPORTANT: greatly improved downloader code and its performances
  - Greatly improved download bandwidth usage
  - Greatly improved disk usage
- CORE: Solved missing Mono certificates validation for HTTPS download URLs
- CORE: Added build installer feature
- CORE: Added build repairer feature
- DOC: New documentation added

v2.0.4:

- GUI: Adjustments to P.A.T.C.H.'s Editor GUI
- CORE/GUI: Upload to FTP added
  - Recursive directory creation
  - Patches and builds upload
- CORE: Added TAR archiving (uncompressed) for patches and deploy
- CORE: Added TARGZ compression for patches and deploy
- CORE: Solved a bug on downloading process for Unity version Launcher

v2.0.3:

- CORE/GUI: One-click-deploy introduced
- GUI: Deploy folder structure created and relative settings added

- CORE: Solved a weird path-relative bug on Mac systems
- CORE: Solved LaunchArg parameter override bug
- CORE: FatalError is now called when archive download fails or archive hash checking fails
- CORE: FatalError is now called after a rollback
- CORE: Cleaning up of single archive file when download fails or hash checking fails
- CORE: Detection of OS related files like ".DS_Store" or "desktop.ini"
- CORE - IMPORTANT: solved OS reliant patch indexer bug, regenerate your patches!
- DOC: Added "Roadmap" in this document
- CORE: Changed CommandLine script to fit with new Unity 5.3 API (about SceneManager)
- CORE: Improved download manager, now it works also on "file://" protocol (so on file system)
  - It offers new information about current download
  - Download progress is now available
  - Download speed is now available
- PROJECT: Added package with Unity 4.6.9 and Unity 5.3.1 to improve compatibility

v2.0.2:

- CORE: workspace cleaning up during patches application

v2.0.1:

- CORE: Solved ZIP compression bug on Mac and Linux systems
- CORE: Solved linear patches application loop
- PROJECT: Added missing localization files
- CORE: Solved three pending warnings

v2.0.0:

- Now Unity integrated
- Localized thanks to Localizatron (it's my localization software)
- All core functions was re-coded
- Patches are now more small
- Changed file hashing algorithm

- New callbacks system added
- New versions standard format

v1.2p1:

- Added a new event to check when current build version changes
- API: PatchManager class now exposes GetCurrentVersion method
- Added a reminder of current build version in Launcher
- Patching process now will apply ever the latest patch available, so if you have 0.1>0.2 and 0.1>0.3, P.A.T.C.H. will apply 0.1>0.3 patch, skipping 0.1>0.2
- FIX: Version constructor now strips correctly “\r” char from remote hashes
- Added some new LogEvents to describe better what is happening during long processing

v1.2:

- Added new in-game GUI that informs your users when checking of launching arguments fails
- Added forced run-to-admin behaviour to Launcher and Patches Builder for Windows
- FIX: now Launcher will apply patches in linear way and non-linear way both, without Launcher restarting
- FIX: added a “s” char in versions.txt example URL, in this way users can't be wrong
- Added support for FTP credentials to download files over Files Transfer Protocol.

v1.1p1:

- FIX: changes to file hashing function is now correctly applied in patching process

v1.1:

- FIX: patching process run now on a new thread
- FIX: delete file process now retry to delete files for a customizable amount (to avoid deadlocks on file deleting)
- FIX: PatchFailed event now doesn't shutdown the patching thread before GUI updating
- Added core version reminder and launcher version reminder to Launcher
- Added core version reminder and patches builder version reminder to Patches Builder

- Added patch rollback feature: if patch process fails all changes will be discarded to avoid a build corruption
- Added generation of patch files indexer in patch building process
- Added hash validation for patched files