M2HPatcher

Mike Hergaarden - http://www.M2H.nl
November 2011

What is M2HPatcher?

A multi platform (Mac + Windows) patch solution for Unity. It allows Unity developers to launch updates of their games in a single mouse click. Just sit back and wait for the automated building, zipping, patch creation *and* uploading of the game files!

Links:
Watch the demo video
Download from the Asset Store
Unity forum thread

Features

- The creation tool is fully integrated in the Unity editor (Mac and Win)
- Supports any Mac and Windows standalone games (plus additional files)
- Full source code included for both the editor scripts and the patch client
- Updates can be enforced&started from inside your game
- Fully automated: just click “Publish” to distribute the latest version to your users.
    1. ..increases the build version number
    2. ..builds a full [Windows, Mac, Webplayer] version of your product
    3. ..creates patches with earlier standalone builds
    4. ..zips the full builds and patch files
    5. ..upload the full builds and patch files
    6. ..publish the latest version number online

DISCLAIMER: Use this package at your own risk. We have tested the package sufficiently in our development environments but we cannot be held responsible for your usage of this package.

The rest of this document is aimed at users who have bought the package. If you are considering to buy this package you can view the rest of the document to see how this package can help you.

Getting started

Let's get you started. To get most use of this packge, you need a webserver where the files and patches can be uploaded, and where we keep track of the current game version.

Requirements:

- Unity PRO (for the automatic building features)
- A server with the following services:
    - An FTP account
    - A webserver (running PHP)
        - You can also use ASP, but you'll have to convert the simple PHP script that has been bundled in this package.

We start by setting up your webserver, after which we connect to it in the M2HPatcher editor settings. The complete setup should take no more then 15 minutes.

Setting up the webserver

The webserver is used to save version information and files. Upload the files from *M2HPatcher/webserver* to your webserver. Make sure to place the files at a directory that is available from the web, we need to enter this URL in the editor settings later. Next, we need to configure the files:

buildversion.txt

This file needs CHMOD 770 so that uploadEditorData.php can write the latest version to this file.

patcher.php

Open this file to modify two links:

a. $*filePath* - the link where your files are available.
b. $*noPatchAvailableLink* - where your users will be pointed to if no patch is available.

uploadEditorData.php

Edit the $*CORRECT_KEY variable* and set this to a random passphrase. Then, also set this same value in the editor patcher settings to ensure no one can easily submit malicious data.

files/

This folder needs to be accessible by an FTP account to upload new builds and patches. Furthermore, it also needs to be available via the web to allow downloading by users. *I advise you to create a FTP account that has access to only this directory since the user/password of this FTP account will be saved in your project folder and are visible to all*

your project users.

This is all you need to do. The server setup has been kept as simple as possible so that you can easily customize the back-end to make it fit your specific needs, if any.

Setting up the editor

For the editor settings, open M2HPatcher->Settings. In this section we go over all the options in this window from top to bottom. You only need to use these settings once, after which you should be able to publish as much as you want.

General settings

Build folder

Best is to leave this to the default value “builds/”. The builds will not be included under asset server versioning on purpose. This is not a problem, since if you miss the last build (because the last publish was done on another machine), the last build will automatically be downloaded from the FTP server.

Build for..

This simple toggle allows you to select your target platforms. Currently windows 32 bit, Mac Intel and streamed webplayer are selected as default.

Edit build version

Version

This is the current development version. This wil always be one version ahead of the live version.

URL:

You need to enter the url to the webserver files here (e.g. http://downloads.m2h.nl/). Do manually test this link. *Mind that this URL should not point to patcher.php or uploadEditorData.php directly!*

Passphrase

Enter the passphrase that you’ve hardcoded in uploadEditorData.php

FTP

The FTP is used to automatically upload the new builds and patches.

Port:

Your FTP server port. 21 per default.

Hostname/IP

The hostname of your server. E.g. M2H.nl *or* the servers IP address.

Username

Your FTP username

Password

Your FTP password

Folder

The folder relative where the game files are stored.

Use FTP

Should the editor use FTP uploading? True per default

Patches only

False by default. If this is enabled only the patches will be uploaded. This can save quite some deployment time.

ZIP level

The compression rate for full-builds (patches are always compressed). This is a balance between your publishing time versus the size of the downloads.
0=off, 1 =worst but fastest, 9=best but slowest compression.

At the very bottom of the settings window there are some extra options:

- *Rebuild patcher clients* allows you to force rebuilding the mac and windows patch clients. Normally these are only built when they are not present (the first time you publish).
- Per default the publish option only makes a patchfile between the current version and the one before. Using the *"Create custom patch"* button you can create a patch between two versions of your choice. You can create a patch between e.g. version 1 and 5. If you upload this custom patch to your FTP *files* folder clients will be able to skip versions 2,3 and 4 because the patcher.php code detects the shortest patch 'route' .
- The *Documentation* button points to this file
- The last credits button directs you to http://www.M2H.n/unity which contains a list of all our Unity resources.

Integrating in your game

Having completed the setup, you can already start using M2HPatcher by usint the PUBLISH button. Users can then use the Patcher application next to your game to update. However, most likely you want to notify users of available patches in-game, so that they don't need to open the patcher. The good news is, M2HPatcher is already integrated in your game via some static classes!

The demo scene demonstrates the use of the M2HPatcher_Runtime class.

```
 void Awake()
{
    //Instantiate the Patcher runtime class.
    // It will automatically check your server for the latest version number.
    M2HPatcher_Runtime.Init();
}

void OnGUI()
{
    //Gets the current build version
    GUILayout.Label("This version: " + M2HPatcher_Runtime.GetThisVersion());
    //Get the onlien version. Will report -1 if it's not yet loaded.
    GUILayout.Label("Online version: " + M2HPatcher_Runtime.GetOnlineVersion());
    //Have we loaded the online version
    GUILayout.Label("Loaded version= " + M2HPatcher_Runtime.HasLoadedVersion());

    // We should only call NeedUpdate if we've loaded the online version
    if (M2HPatcher_Runtime.HasLoadedVersion())
    {
        //Is the online version > current version? Then NeedUpdate() is true.
        if (M2HPatcher_Runtime.NeedUpdate())
        {
            //We need to update: You can FORCE this or ask the user to update.
            if (GUILayout.Button("Update me!"))
            {
                //Quit game and start Patcher application
                M2HPatcher_Runtime.QuitAndStartPatcher();
            }
        }
    }
}
```

The game installers

Adding files next to the game
The current M2HPatcher creates zip files that include the full game. In this zip file is your game, the patch application and optional content, such as documentation, that you can place in *M2HPatcher/BundleWithGame*.

An installer instead of a zipfile?
Ideally, windows installers should be .MSI files and Mac installers DMG. You can create these installers manually or extend upon M2HPatcher to do so. Creating DMGs can be as simple as drag&dropping the contents of the full build zipfiles into a DMG create application.
Creating a MSI is also only 3 minutes drag&drop work, see our blogpost: http://blog.m2h.nl/2011/03/creating-msi-installer-for-unity-games.html. While the blogpost is qutie length, if you've done it once it's dead easy.

M2HPatcher does not automatically create MSI and DMG installers as we have not yet found a method to do so multi-platform.

F.A.Q.

What platforms does M2HPatcher support?
Currently M2HPatcher allows you to select building for Windows32, MacIntel and Webplayers. M2HPatcher will only create patches for standalone builds (Mac, Windows, *Linux in the future?*). You can modify the code to use different standalone builds such as Windows64 or MacUniversal.

Patching Android/iOS/Xbox/Wii builds makes no sense as the platforms have their own update systems. Do note that you are also not allowed to include your own update functionality when adding games to the Mac game store. The same will probably count for platforms such as Steam.

What does the menu option "M2HPatcher->SingleBuild->.." do?
The single builds allow you to create a single build for the option you selected. No patches are made , no zipping is done and no version is changed at all. These builds use the current development version and will be placed in the *builds/* folder. Note that these manual builds will be overwritten as soon as you use the publish button as then the final versions will be build. The SingleBuild option is handy if you quickly need a standalone (development) build.

How can I customize the patch client?
Simple browse to "*M2HPatcher\Files\PatchApplication*" and customize the patchscene. You can customize it however you like, as long as you make sure not to break the functionality in the *PatchClientGUI.cs*.

Why do you use a Unity application for the patchclient / What is the overhead of the Unity patchclient?
Compressed (zip) the patcher is 8.3mb on windows and 14.4mb on Mac. This is quite some overhead for a patch client. However, we did this on purpose: We want to offer the easiest integration in Unity so that you can easily customize the client. You can reuse the patch client code to create a dedicated .NET application to save some file size here.

How to integrate M2HPatcher with my own build system?
If you want to use your own build solution you'll want to re-use only the Patch creation of M2HPatcher and the patch client. The patch client can be easily copied next to your builds, as it's an entirely separate application (adapting the webserver will be trivial).

To rip M2HPatcher apart you can start by looking at the Publish() function of M2HPatcher**.** You'll find that the key patch creation functionality is done in the M2HPatcher_CreatePatch.MakePatchForDir function. This function creates .m2hpatch files that are used by the patch client.

What files does the patcher support?
The patch client supports all kinds of files.

How does it work in a team?
M2HPatcher is compatibly with the Unity asset server (or SVN etc.). There are two setting files inside the asset folder (SettingsDev and SettingsDeploy). The only thing you need to keep in mind that if one publishes, these builds are saved to the local builds/ folder, which is not in the project folder per default. To make this less of a problem, if you're missing the last build M2HPatcher will attempt to download the last build from the FTP server and use that to create patches against.

What about future support/updates etc.?
You buy this packase as-is, we have no plans for future updates. We are available for questions and will do our best to offer a reasonable amount of support on this product. We will of course be releasing bugfix updates where required.

You can of course extend M2HPatcher for your own needs, if you feel like sharing your improvements we'd gladly add them to the main product.

While you should not count on the following features being implemented, this is our "internal" wishlist/TODO list for M2HPatcher:

- Better fallback for patcher errors (display custom text when offline, Totally redownload game or patcher)
-
- Specify to automatically make patches for X versions back instead of always only the last version.
- Customize naming of the game builds.
- Installers instead of zips for the full games (MSI and DMG)
- Webserver(PHP) should be able to tell if a patch is mandatory
- Webserver(PHP) should allow a full-redownload of the game if patching somehow messed up
- Add more build management support
    - Targets: Support multiple builds per platform (standalone game, standalone demo etc.)
    - Customize the scenes per target
- You cannot yet move/rename the M2HPatcher folder. (Is this required?)
- PRE/POST publish delegates?
- Add multiple FTP servers?

Known limitations

Renaming the application on Mac will stop the application from working
Renaming your game on Windows is no problem. However, renaming the application causes the files to be created from code, with no execution rights on Mac. To fix this on Mac, chmod +x needs to be run on YOURGAME.app/Contents/MacOS/YOURGAME. This can possibly be done from code by executing this from the patcher.

A workaorund that I've used in the past is quoted below. However, a nicer solution will be added to M2HPatcher when available.

> "The current workaround is to ship the patcher with 'FixPermissions.app'. After an update the patcher would call:"

```
System.Diagnostics.Process Proc = new System.Diagnostics.Process();
Proc.StartInfo.FileName = "FixPermissions.app/Contents/MacOS/applet";
Proc.Start();
```

> "However, it seems that the application is not (always) executed. Therefore a custom user notification to run the application is advised instead.
> Hereby the AppleScript sourcecode to create your own FixPermission.app:"

```
on run input
	set theDirection to "Cubelands.app:Contents:MacOs:Cubelands"
	tell application "Finder"
		set my_folder_path to container of (path to me) as text
	end tell
	set clPath to POSIX path of (my_folder_path & theDirection as string)
	do shell script ("chmod +x " & clPath)
	get "DONE: " & clPath & " Dir:" & theDirection
end run
```