TexturePacker(Pro) Manual

Table of Contents

Overview

TexturePacker is a command line tool to create sprite sheets or sprite atlases without manual interaction! Without specifying any additional options TexturePacker already creates very good results.. But you also have enough options available to adjust things to your needs.

Main features:

- Graphical user interface
- Free .pvr and .pvr.ccz previewer
- PVRTC compression
- Full automatic operation
- Can be integrated in your build process
- Optimized output
- On-The-Fly rescaling
- Robust
- Fast
- Extruding sprites for flicker free tile maps
- Automatic creation of aliases for identical images
- Smart folders for simpler updating
- Drag'n'drop sprites from finder
- Xcode integration

TexturePackerPro vs TexturePacker

TexturePackerPro and TexturePacker are the same application. The only difference is that the Pro version has GUI support. Standard TexturePacker can start the GUI but all files saved will have some Sprites turned to red when exporting.

Supported operating systems

TexturePacker currently runs on MacOS 10.6 and above. (10.5 may work too but since I have no system for testing it is not recommended).

Every release version will be built for Windows XP, Vista and 7. Beta versions are built on demand for Windows.

Linux build are made on demand.

Installation

MacOS

To install TexturePacker simply start the TexturePacker.pkg. It will install the main program in Applications and a link for easy use in `/usr/local/bin/TexturePacker`.
TexturePacker and TexturePackerPro are identical software. The license destinguishes between both versions. If you own a TexturePacker license you can test the graphical user interface in demo mode.
TexturePacker licenses can be upgraded to TexturePackerPro.

Windows

To install TexturePacker on Windows simply run the installer. TexturePacker is available in 2 exectuable files: TexturePacker.exe which is a command line tool (please add it to your path if you want to use it) and TexturePackerGUI.exe which is the graphical user interface.

Essential mode

TexturePacker runs in essential mode by default. In essential mode basic sprite sheet functions are available but not extended features like image optimization or other algorithms than Basic.

Installation of a license file

To install the license file open a shell and execute:

```
TexturePacker --install-license <name_of_the_license_file>
```

The license file will be copied to

`/Users/<user>/Library/Preferences/de.code-and-web.TexturePacker.plist`

The license can also be installed using the Graphical User Interface. TexturePacker prompts for a license if none is installed. You can also go to the Preferences menu to enter a new license.

Quickstart (TexturePackerPro, GUI)

Click the TexturePacker icon in your Applications folder.

Quickstart (shell)

Open a shell and use this command to create a sprite sheet in fully automated mode from all png files in the assets folder:

```
TexturePacker assets
```

TexturePacker reads all png files in the assets folder and creates the texture with the minimum area using all its default settings. Output is written to `out.png` and `out.plist`.

Help (—help)

You can get a list with all commands and information by using --help or simply calling TexturePacker with no arguments.

If a command in this documentation does not work (due to e.g. a typo) please check with --help.

Edit with GUI, export from Commandline

Users of TexturePackerPro can edit their sheets and save the settigns as *.tps files. You can now update the sheets from command line by calling

```
TexturePacker *.tps
```

This call will update all texture and data files set in the tps files. File globbing is currently not shpported under Windows. You must name all files to be processed.

You can also set options which will override the options set in the *.tps file. E.g. for exporting the same sheet with different scale or color optimization.

Batch conversion of sprites

To convert a bigger amount of images without creating sprite sheets you can use the command line with the following parameters:

```
find <directory> -name *.png -exec \
       TexturePacker {} --sheet {}-converted.png \
           --no-trim --disable-rotation --padding 0 <params> \;
```

Please replace <directory> with the directory to search and <params> with additional parameters.

For PNG export you might also want to set --allow-free-size to preserve the file's original size.

Options

Specifying a layout algorithm (—algorithm)

The default algorithm for TexturePackerEssential is Basic, TexturePacker(Pro)'s default is MaxRects.

Basic

Basic is a simple algorithm just lining up sprites, starting a new line if one is full. This layout algorithm is designed for tile maps and users for which a simple algorithm is sufficient.

Options are --basic-sort-by which allows sorting by name, width, size, area or circumference. The sort order can be chosen by **--**basic-order, valid values are ascending, descending.

By default the best fitting sorting is chosen.

MaxRects

MaxRects is one of the best layout algorithms. It is fast and achieves a high sprite density. Heuristics for setting the sprites can be chosen with --maxrects-heuristics but usually the default is sufficient: TexturePacker choosed the heuristics which produce the minimal texture size.

Specifying input files

To specify the input files you can simply add file names. Everything that is not a known parameter will be opened as file.

You can use your shell's typical file globbing to make adding files easier, e.g. *.png for all png files (MacOS only).

See supported image formats below for a list of supported file types.

Smartfolders

If you specify a folder the contents will be scanned recursively for png files. The sub directories will add to the sprite's name.

This allows organization of sprites within folders – e.g. for animations etc.

- assets
  - ball
    - bounce
      - 01.png
      - 02.png
    - shrink
      - 01.png
      - 02.png

If you add the assets folder sprites will be named `ball/bounce/01.png` etc.

Ignoring image files (—ignore-files <regexp>)

You can use a wild card here to ignore files matching the expression. „*“ and „?“ are allowed. Make sure to escape them so that bash does not already match files.

Removing .png and other file extensions from sprite names (—trim-sprite-names)

This option removed the file name extension from the sprite, e.g. .png

Renaming (parts) of sprite names (—replace <rx>=<str>)

Replaces matching parts of the regular expression <rx> in the sprite's name with <str>. Please make sure to escape the regular expression so that bash does not already match files.

Setting output images format and name(—sheet <name>)

By default the sheet is written to out.png and the plist file to out.plist. You can specify other locations with –sheet and –plist.

By choosing a different ending you can also output other image formats.

Supported formats are:

- .PNG
- .PVR – iPhone optimized image format
- .PVR.CCZ – compressed format for cocos2d (currently available in develop-branch)
- .JPG use --jpg-quality <value> to set the quality. JPG may require setting high padding values to get rid of artefacts.

Chosing output data format (—format <name>)

Currenty one formats are supported:

cocos2d

plist file for cocos2d iphone

corona

Lua file for Corona (TM) SDK

sparrow

XML file for Sparrow Framework

libgdx (beta)

Text file for libGDX

Update sheets only if sprites change (—smart-update)

With this option TexturePacker checks file sizes, file names (missing, added or renamed files), modification dates and parameters. If TexturePacker is run and nothing changed since the last run the build process is just skipped. Use --smart-update to enable this feature.

This is useful for integration into build tools like Xcode.

Controlling sprite sheet size (—width, —height, —max-*)

By default TexturePacker uses its algorithm to find the minimum area which means the least memory to use for the sprite atlas.
Without any parameters set TexturePacker starts from a minimum size of 1 increasing in powers of 2 until maximum width or maximum height is reached (which is 2048 by default)

You can control the output by setting maximum values for width and/or height or by directly setting the value.

--max-width <int>	Sets the maximum width of the sprite sheet to the given value. The width may be smaller if a smaller size is found. Default is 2048.
--max-height <int>	Sets the maximum height of the sprite sheet to the given value. The height may be smaller if a smaller size is found. Default is 2048.
--max-size <int>	Is a short hand, setting both maximum width and height.
--width <int>	Fixes width to the given value. May be any positive number.
--height <int>	Fixes height to the given value. May be any positive number.

Creating non power of two textures (—allow-free-size)

The layout still tries to optimize for power of two sizes to reduce memory on the target device but this option creates minimum texture sizes for PNG and JPG export with smaller sizes if possible.

This helps creating simple sheets with for animations without the use of an additional data file.

Resizing sprites on the fly (—scale <int>)

One very powerful feature of TexturePacker is the on-the-fly resizing. You can create all your assets in high resolution and scale the sprites down during the creation process.

Use --scale 0.5 to resize all sprites to 50%.

This creates 2 sheets: One with up to 2048 pixels for high resolution and another with 1024 pixels maximum width with all sprites scaled down to 50%:

```
TexturePacker assets/*.png --data hd.plist --sheet hd.png
TexturePacker assets/*.png --data sd.plist --sheet sd.sheet \
                           --resize 0.5 --max-size 1024
```

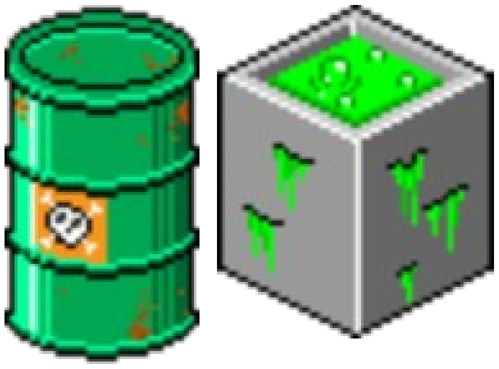
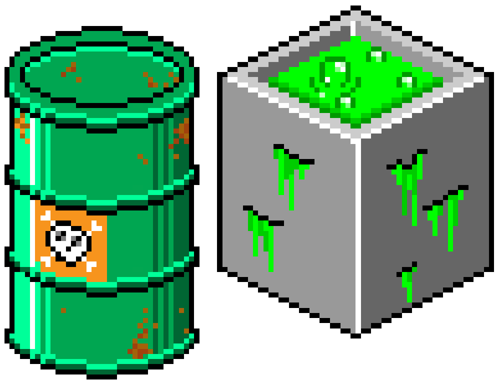
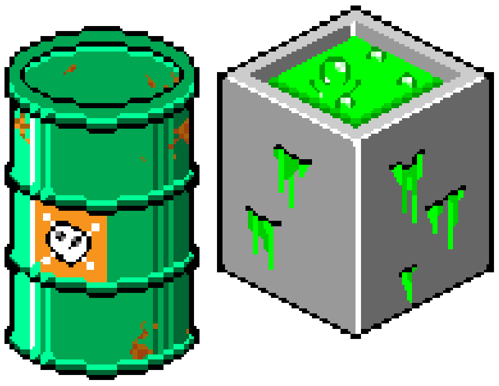
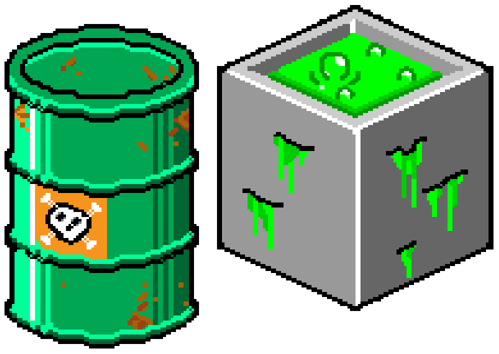
Choosing scale mode (—scale-mode <int>)

Scale mode allows you to control how scaling is done. By default smooth scaling (which is bilinear filtering) is activated. TexturePacker also has a nearest neighbour mode (called fast). Both are general purpose scaling modes working with any scale factor for up- and downscaling.

TexturePacker also has special surpose scaling modes for upscaling by a constant factor 2.
These modes are:

- eagle
- scale2x
- hq2x

See http://en.wikipedia.org/wiki/Pixel_art_scaling_algorithms
The examples below show upscaling with the diffent algorithms. Since the quality in this document might not be the best simply try them in TexturePacker.

ScaleMode (2x)	Result
smooth (Bilinear filtering)	
fast (Nearest neighbour)	
eagle	
scale2x	

hq2x	

(Thanks to ITL Games http://www.itlgames.com for the images!)

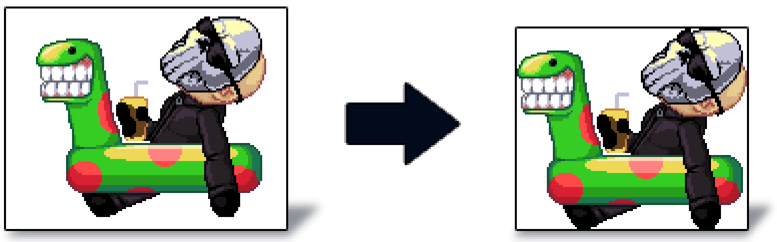
Creating hd and sd sheets in one run (—auto-sd)

TexturePacker asumes that the assets are in high definition and requires texture and data names to end with -hd or @2x.

It now exports the sheet with -hd / @2x ending at the set scale and additionally one sheet with 0.5 scale without the ending.

Trimming (—no-trim, —trim, —trim-threshold <int>)

Trimming removes transparent parts of the sprites. This allows a higher sprite density in the texture atlas and faster drawing of the sprites since the transparent pixels are not drawn.
Trimming preserves the original sprite size in the data file.
By default trimming is on, use --no-trim to deactivate it.
--trim-threshold allows settings the threshold for pixel's transparency. This allows cropping sprites which have nearly invisible pixels.

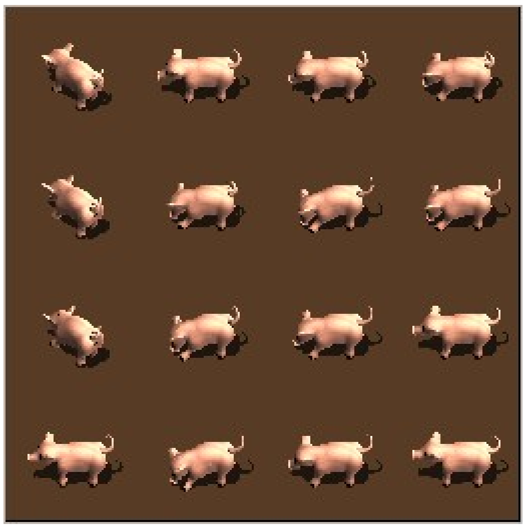


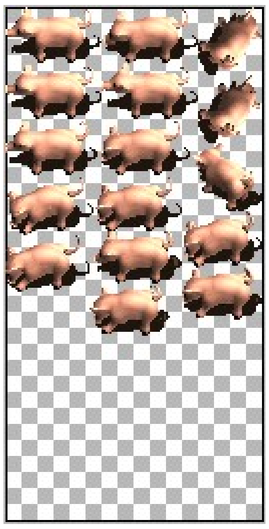
Cropping (—crop)

Cropping is similar to trimming except that the sprite's size in the data file is exported as the reduced sprite.

Creating heuristic masks (—heuristic-mask)

This feature helps creating sprite sheets from sprites which use a single color to mark transparent values.



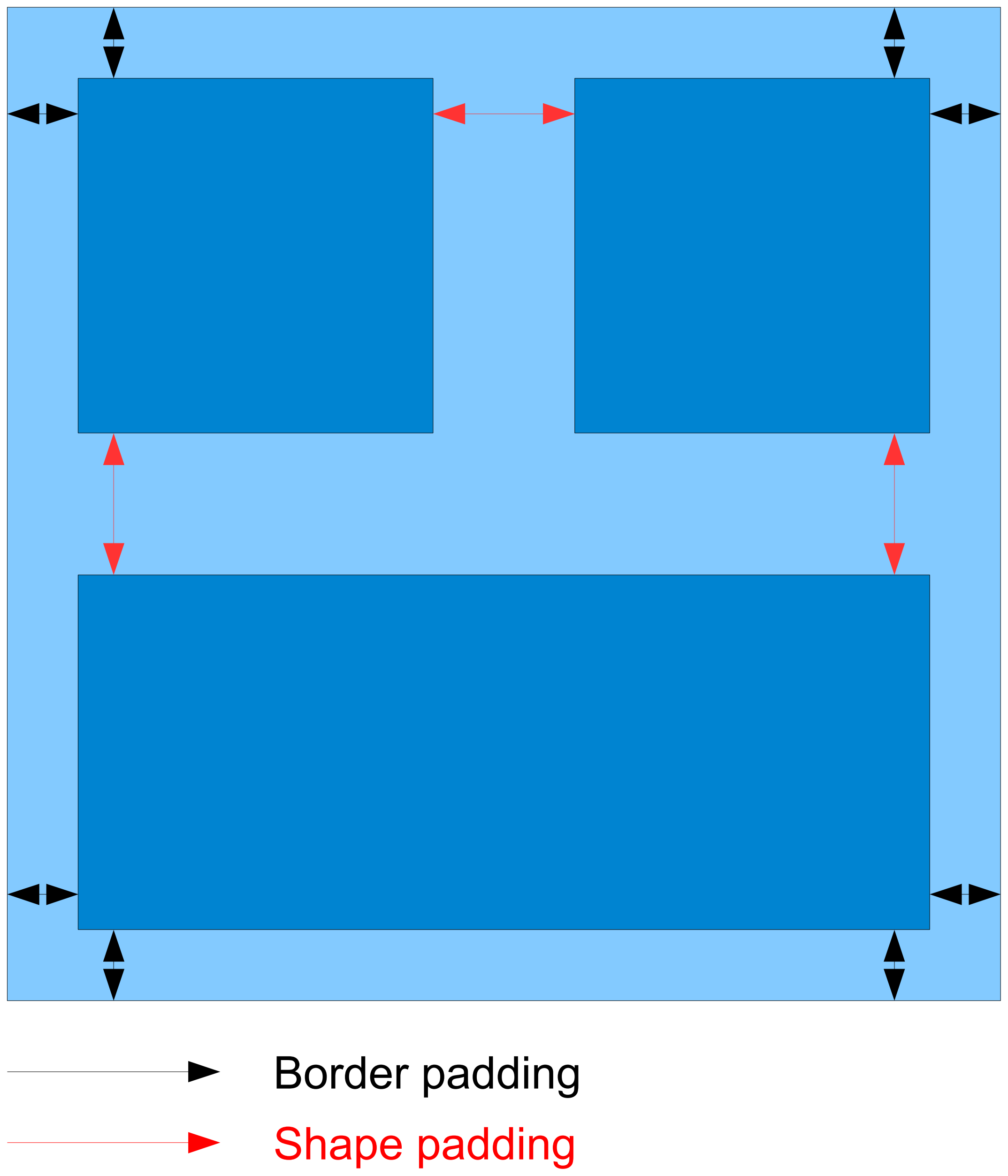


Rotation (—enable-rotation, —disable-rotation)

This option disables / enables rotating sprites. Enabled rotation may create smaller sprite sheets.

For the plist file rotation is enabled by default. You can disable it to use it with older versions of cocos2d by using –disable-rotation.

Padding



Shape padding (—shape-padding <int>)

Adds space between the sprites. 0 by default.

Border padding (—border-padding <int>)

Adds space between the sprites and the border. 0 by default.

Padding (—padding <int>)

Adds space around all sprites and to the border. Short hand for --shape-padding and --border-padding. By default padding is deactivated to create the smallest possible size of texture.

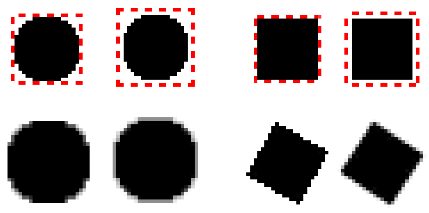
If you want to use the texture in repeat/tile mode it is better to set --shape-padding to a value and --border-padding to the half of it. This is because in repeated mode left and right side add the double of the border padding.

Padding (—inner-padding <int>)

Adds transparent pixels around the sprite. In contrast to the other padding options this enlarges the sprite.

There are two uses for this:

- It can help in preventing cut-off artifacts near the edges of scaled sprites. E.g. if your sprite has a few pixels along its own boundaries, scaling the sprite up or down won't let these pixels appear like gaps or cuts.
- It considerably reduces aliasing along the polygon edges when rotating trimmed or cropped sprites. E.g. if your sprite has many pixels along its own boundaries, it will be drawn more smoothly when rotating it."



Optimizations (—opt <type>)

TexturePacker can reduce the output size of the sprite sheet by reducing the color of the created sprite sheets. In some cases this results into a dramatic reduction of file size. The following pixel formats are available for export:

- RGBA8888
- BGRA8888

- RGBA4444
- RGBA5555
- RGBA5551
- RGB565
- PVRTC2 (lossy compression)
- PVRTC4 (lossy compression)
- Alpha

--opt RGB565 discards all transparency information. You can use --background-color <rrggbb> to set the background.
Alpha creates a black and white image of the sprite's alpha channel. Black means transparent, white opaque.

Premultiplied alpha (—premultiply-alpha)

Several frameworks (e.g. cocos2d) support/require premultiplied alpha – especially for exporting .pvr and .pvr.ccz
This helps reducing artefacts at the sprite borders under cocos2d when texture antialiasing is enabled. Should not be activated for PNG files.

Dithering (—dither-*) and color quantization

Dithering enables dithering while reducing colors. The advantage is improved color matching but the images get a bit grainy. This is ok for moved sprites but may be disturbing for non moving images.

--dither-none-nn does a simple nearest neighbor color matching. It creates a smaller color error than --dither-none-linear but also has less contrast

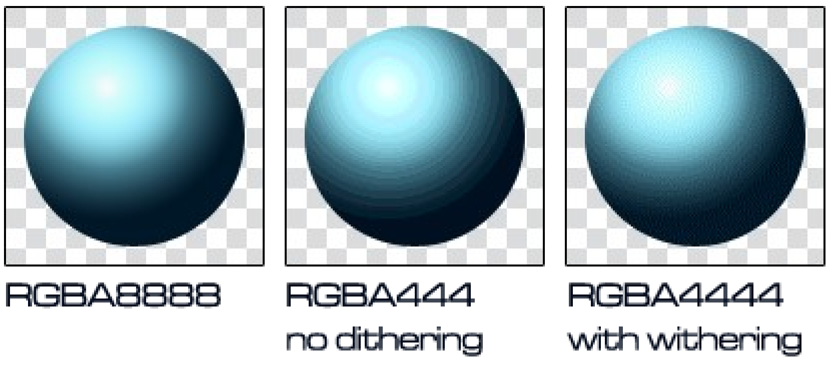
--dither-none-linear does a linear color matching, accepting a bigger color error than –dither-none-nn but preserves contrast

--dither-fs only dithers color channels and preserves transparency. Use this function to keep sprite borders crisp or when working with sprites which are used as tiles

--dither-fs-alpha also dithers alpha channel. Use this if your sprites use soft alpha blending.

--dither-atkinson uses atkinson dithering

--dither-atkinson-alpha atkinson dithering with alpha channel



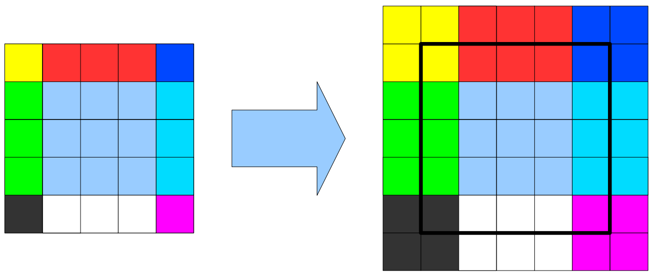
Setting a non transparent background color (—background-color <rrggbb>)

Replaces the transparent background color of the sprite sheet with the given color. This option is usually only used when creating RGB565 optimized textures.
The color value consists of 3 pairs of hex digits representing each color channel. Order of the channels is red, green, blue.
Examples

red	`ff0000`
green	`00ff00`
blue	`0000ff`
black	`000000`
white	`ffffff`

Extruding (—extrude <int>)

Extruding creates a border around the shape with the same pixels repeated from the border. This can be used to reduce flickering in tile maps. Default is off.
The shape's core size stays the same. This can also be combined with padding.



Settings JPG quality (—jpg-quality <value>)

Use this option to change the quality when exporting jpg textures. -1 for default, 0..100 where 0 is low quality.

Flipping PVR images (unity framework) –flip-pvr

PVR Textures can be flipped using –flip-pvr

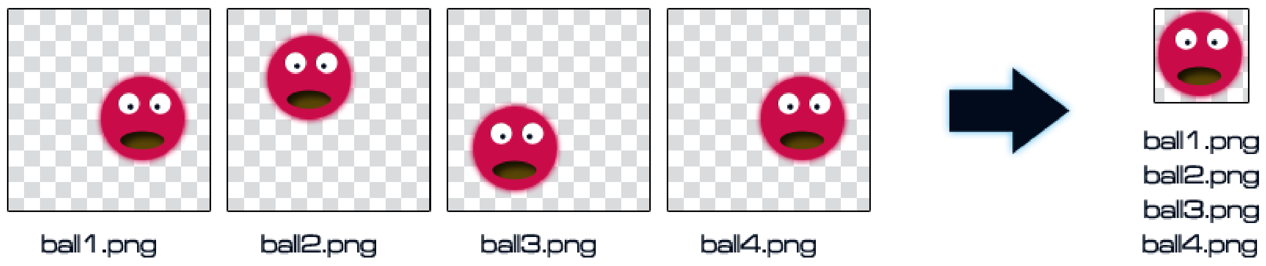
Testing sprites (—shape-debug)

Will draw semi transparent rectangles around the sprites area. Red rectangles represent rotated sprites, green represent non rotates sprites.

Other features

Aliases

TexturePacker automatically creates aliases for sprites which are the same after optimization. E.g. if use an animation which has the same images in some phases – but perhaps just shifted – TexturePacker will only save one version of the image:



Disabling aliases (—disable-auto-alias)

This option disables alias creation. This option was added to keep all animation phases and tiles when using Basic layout algorithm where position of single sprites matters.

Xcode-Integration

The main advantage of TexturePacker is that it can be directly integrated into Xcode. So you don't have to care about updates anymore.

For his short instruction following assumptions are made:

- Sprites are in the project directory under assets
- The sheet will be created into the resources directory

TexturePackerPro

With TexturePackerPro it is now possible to updated *.tps files directly from your build toolchain or XCode. Edit *.tps files in GUI mode and save them.

This script is also available from the scripts folder. If you are a pro user and want to use the automated updates copy scripts/PackTextures-pro.sh and rename it to PackTextures.sh

If you add more options to the TexturePacker call these options will override the settings in the tps files.

```
#! /bin/sh
TP=/usr/local/bin/TexturePacker
if [ "${ACTION}" = "clean" ]
then
     # remove sheets
     rm ${SRCROOT}/resources/sheet*.png
     rm ${SRCROOT}/resources/sheet*.plist
else
     # create all assets from tps files
     ${TP} *.tps
fi
exit 0
```

Commandline

Create the script to update the sheet as scripts/PackTextures.sh. The file is available on the disk image and can be copied from there.

```
#! /bin/sh
TP=/usr/local/bin/TexturePacker
if [ "${ACTION}" = "clean" ]
then
     # remove all files
     rm ${SRCROOT}/resources/sheet.png
     rm ${SRCROOT}/resources/sheet.plist

     rm ${SRCROOT}/resources/sheet-hd.png
     rm ${SRCROOT}/resources/sheet-hd.plist
else
     # create hd assets
     ${TP} --smart-update ${SRCROOT}/assets/*.png \
          --format cocos2d \
          --data ${SRCROOT}/resources/sheet-hd.plist \
          --sheet ${SRCROOT}/resources/sheet-hd.png

     # create sd assets from same sprites
     ${TP} --smart-update --scale 0.5 ${SRCROOT}/assets/*.png \
          --format cocos2d \
          --data ${SRCROOT}/resources/sheet.plist \
          --sheet ${SRCROOT}/resources/sheet.png
fi
exit 0
```

This sheet creates a sheet.png and sheet.plist for sd and sheet-hd.png and sheet-hd.plist from all png files in assets.

The important option is --smart-update – this option only created the sheets if the source files are newer.

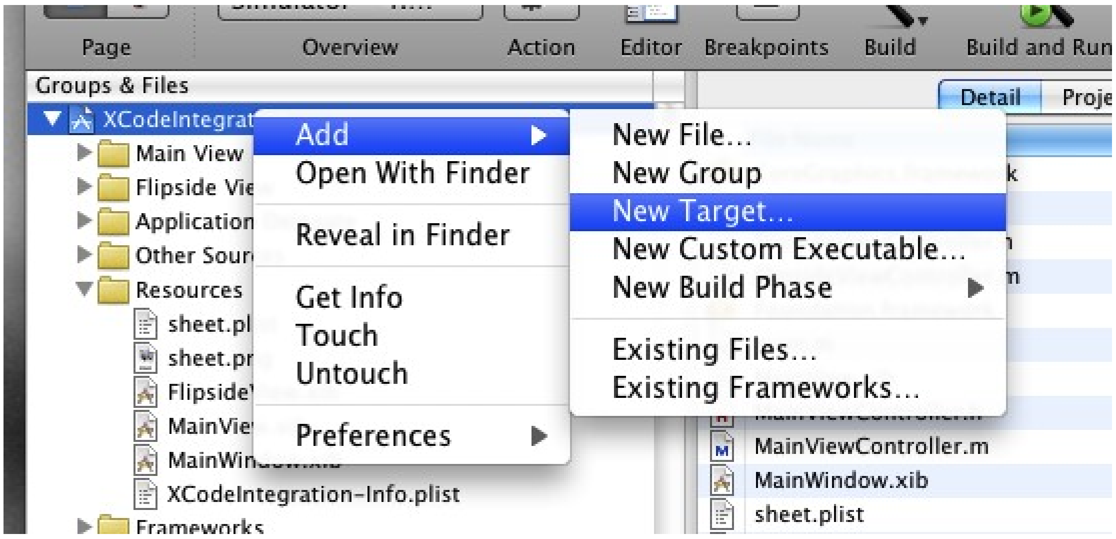
The „cleaning“ part is used if you clean all targets, the building updates the textures.

TexturePacker automatically checks if any of the source files is newer and only in this case the files are updated.

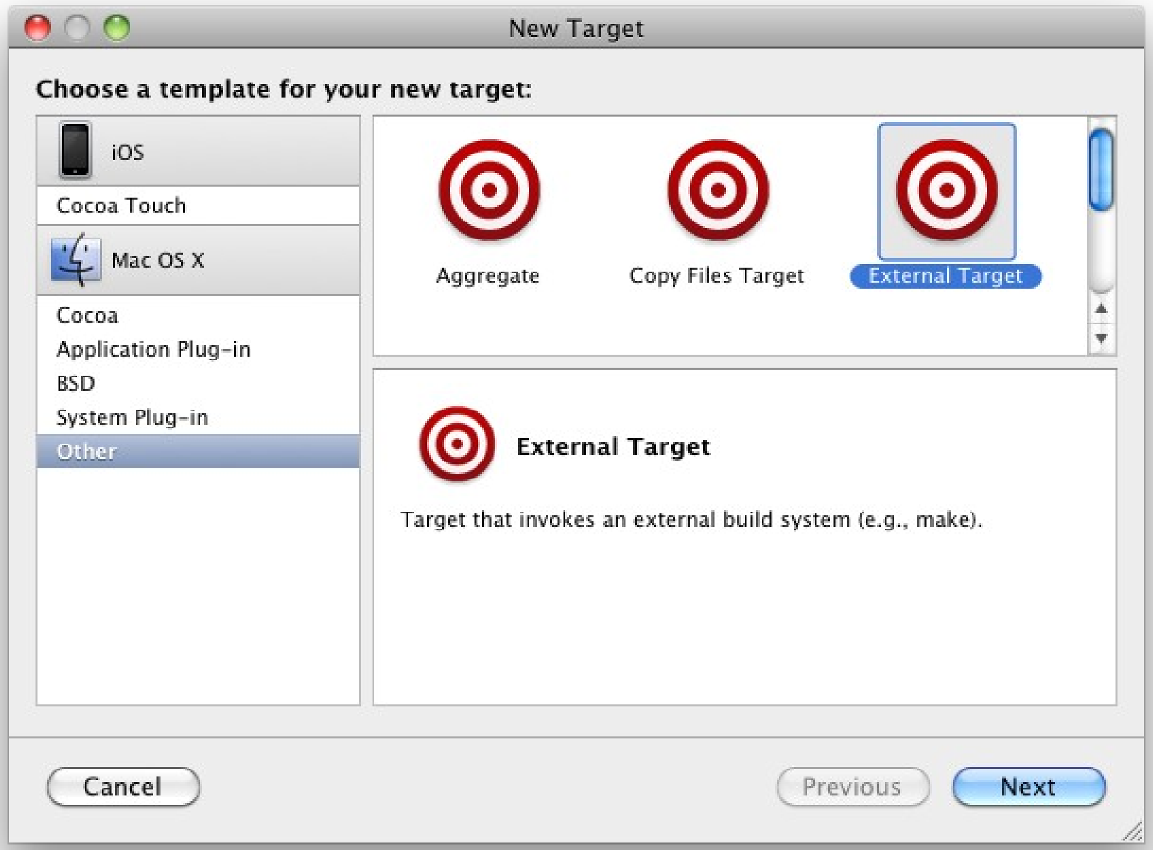
You can add multiple sheets into one simple file.

Tipp: If you add the sheet to the Xcode project as source file you can edit it directly without leaving Xcode!

Add a new target:



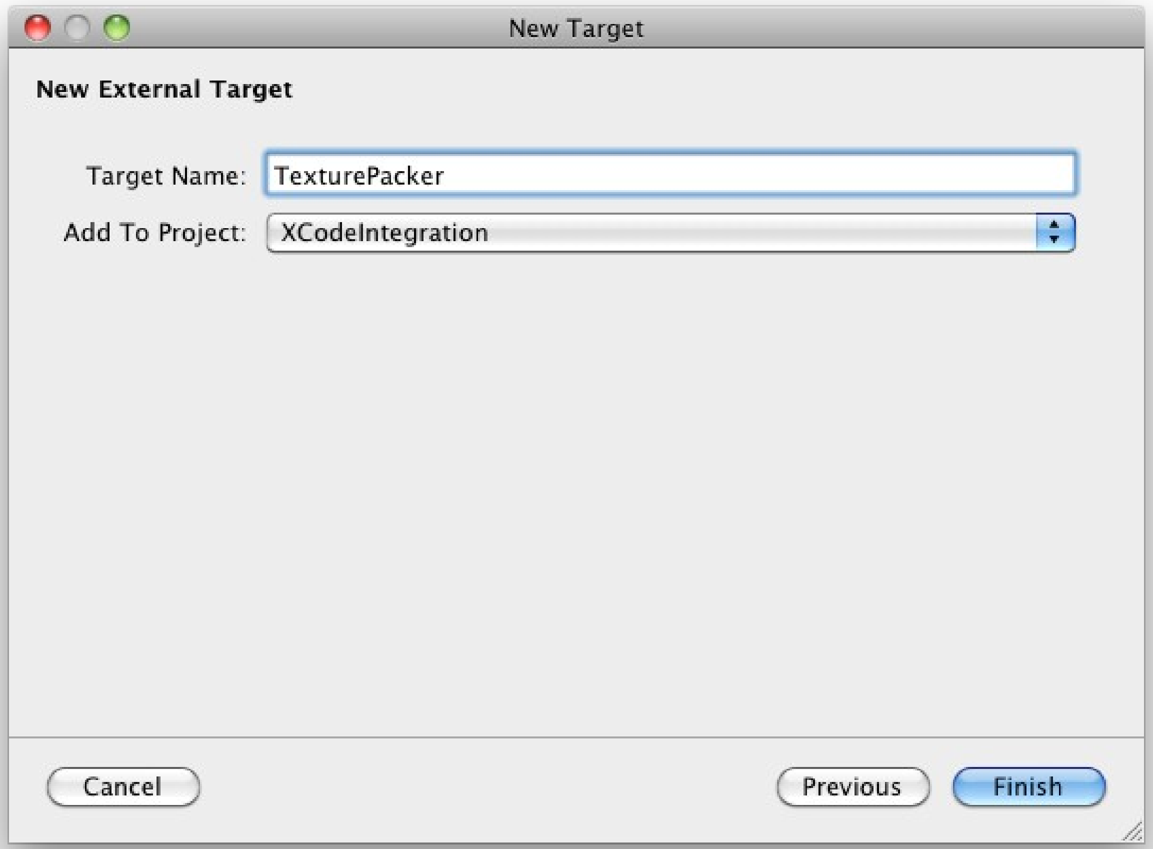
Make an external target:



External Target

Target that invokes an external build system (e.g., make).

Call the target TexturePacker

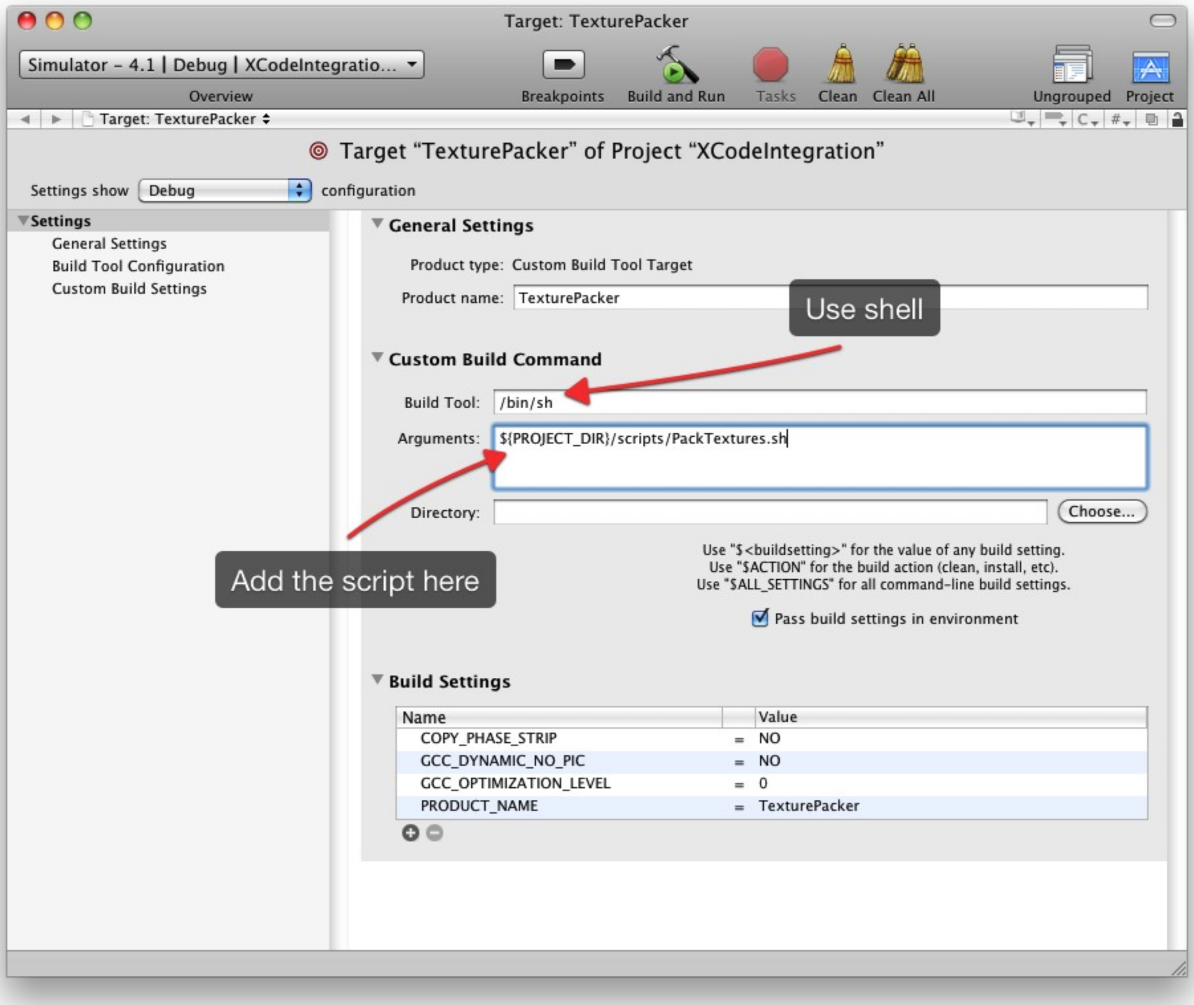


New External Target

Target Name: TexturePacker

Add To Project: XCodeIntegration

Add our script:



Target "TexturePacker" of Project "XCodeIntegration"

Settings show Debug configuration

Settings
- General Settings
- Build Tool Configuration
- Custom Build Settings

General Settings

Product type: Custom Build Tool Target
Product name: TexturePacker

Custom Build Command

Build Tool: /bin/sh
Arguments: ${PROJECT_DIR}/scripts/PackTextures.sh
Directory:

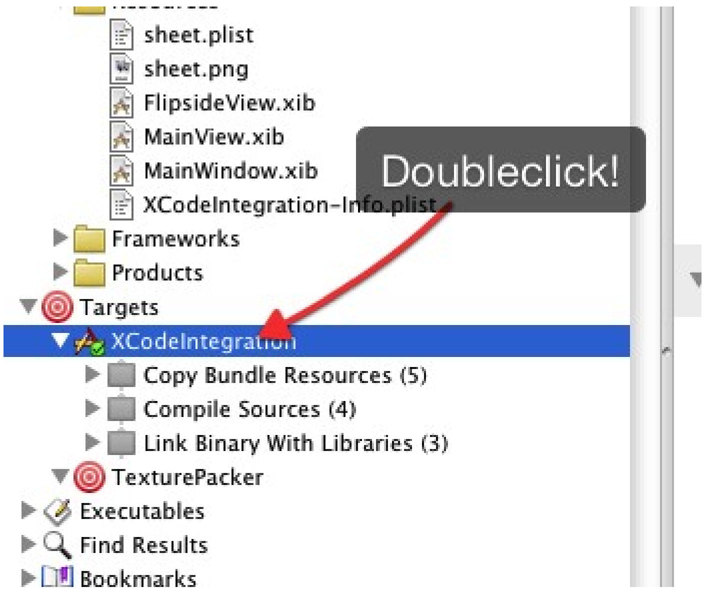
Use "$<buildsetting>" for the value of any build setting.
Use "$ACTION" for the build action (clean, install, etc).
Use "$ALL_SETTINGS" for all command-line build settings.

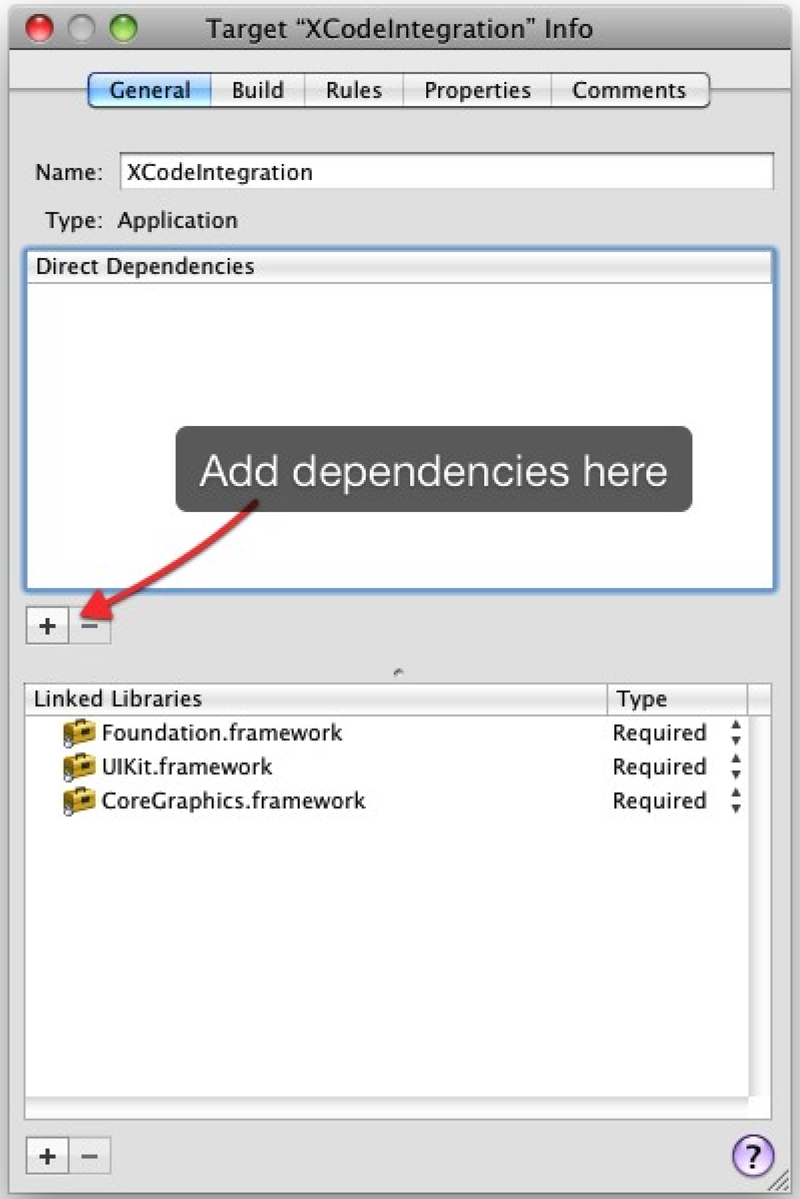
[✓] Pass build settings in environment

Build Settings

Name		Value
COPY_PHASE_STRIP	=	NO
GCC_DYNAMIC_NO_PIC	=	NO
GCC_OPTIMIZATION_LEVEL	=	0
PRODUCT_NAME	=	TexturePacker

Now we have an external target but we need to add dependencies from the main target so that the Textures are build before:





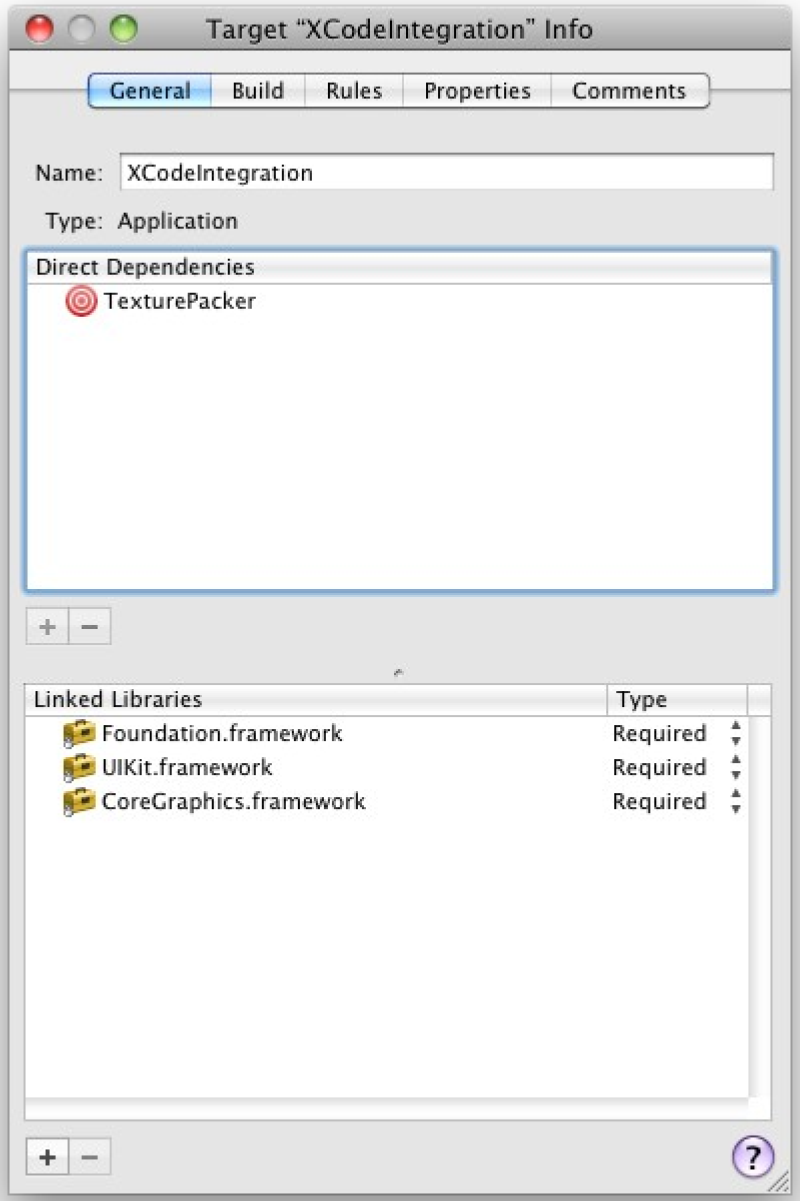
Name: XCodeIntegration

Type: Application

Direct Dependencies

Linked Libraries	Type
Foundation.framework	Required
UIKit.framework	Required
CoreGraphics.framework	Required

Add the TexturePacker as dependency.



Name: XCodeIntegration

Type: Application

Direct Dependencies

TexturePacker

Linked Libraries	Type
Foundation.framework	Required
UIKit.framework	Required
CoreGraphics.framework	Required

That's all.

Starting a new build will now update the textures.

Supported image formats

TexturePacker supports the following input and output formats:

- png
- gif
- jpeg
- tiff
- pvr
- Zlib compressed pvr (pvz, pvrz)

Image formats are automatically applied depending on the ending of the file name specified.

.pvz / .pvrz in detail

This format is simply a zlib compressed .pvr file prepended with 4 bytes size of the decompressed file.

Test sprite sheet is 512x1024, RGBA4444				
File format	Size (in bytes)	Loading times (in ms)		
		iPhone 3G	iPhone 3GS	iPhone 4
PNG	485824	409	165	128
PVR	1048628	81	32	25
PVZ	271758	146	66	47

Test sprite sheet is 512x1024, RGBA8888				
File format	Size (in bytes)	Loading times (in ms)		
		iPhone 3G	iPhone 3GS	iPhone 4
PNG	724090	471	192	155
PVR	2097204	150	60	44
PVZ	631902	283	119	93

```
#include <zlib.h>
...
const unsigned char *data = <pointer to pvz data in ram>;
int dataLen = <size of pvz file>;
...

// extract length
uLongf len = (data[0] << 24) | (data[1] << 16) | (data[2] << 8) | data[3];
data+=4;

// allocate memory
unsigned char *decompressed = malloc(len);
if(!decompressed) {
        // no memory
}

// decompress
uLongf destlen = len;
if(Z_OK != uncompress(decompressed, &destlen, d, dataLen-4)) {
        // decompression failed
}
```

Submitting bug reports

To make handling simpler you can create a debug information file using the options you found the bug with and adding –create-debug-info
This creates a debug information file (debug_info.bin) which you can send to support@code-and-web.de
Please add a short description of what you think is wrong and what you expect to happen.

The file contains

- Version information about TexturePacker
- All arguments you called it with
- Sizes and a black/transparent version of all the shapes
  (This is a convenience function for you so you can submit the data without being afraid.)

The file is compressed but you can peak inside the debug information file with –extract-debug-information <filename>.

By the way: code-and-web respects copyright so all data you transmit is only used for debugging and testing TexturePacker. We do not use submitted data except for this purpose.

Uninstall

TexturePacker installs the following files on your system:

- TexturePacker.app in /Applications
- TexturePacker link in /usr/local/bin/TexturePacker

If you install a license the license file will be located under

- /Users/<user>/Library/Preferences/de.code-and-web.TexturePacker.plist

Thanks

Thanks to the beta testers:

- CJ Hanson of Hanson Interactive (www.hansoninteractive.com)

Also thanks to Andy Matuschak for his great Sparkle framework for automated software update!

Other stuff

Corona TM SDK is a trademark or registered trademark of Ansca Inc.
Ansca the Ansca Logo, anscamobile.com are trademarks or registered trademarks of Ansca Inc