TexturePacker Tool Primer (Version 3)

1. Overview

Purpose: TexturePacker is a command-line utility designed to bundle multiple individual image files (like PNG, JPG, GIF) into a single, optimized binary file with the .xbt extension. This format, specifically **XBTF v3**, is used by the Kodi media center for its skinning engine (starting with version 22 "Piers").

Primary Benefit: Bundling textures improves performance by reducing the number of individual file I/O operations, leading to faster skin loading times. The tool also performs several key optimizations, including duplicate detection, image format conversion, and compression.

Core Functions:

1. **Packing:** Recursively scans an input directory for supported image files and packs them into a single .xbt file.
2. **Information Display:** Reads an existing .xbt file and prints detailed information about its contents to the console.

---

2. Command-Line Usage

The tool's operation is controlled by command-line flags.

Flag(s)	Argument	Required?	Description
-help, -h, -?	(None)	No	Displays the built-in usage information and exits.
-input <dir>, -i <dir>	Directory Path	**Yes** (for packing)	Specifies the root directory containing the image files to be packed. The tool will scan this directory and all its subdirectories.

-output <file>, -o <file>	File Path	No (for packing)	Specifies the path and filename for the output .xbt file. **Default: Textures.xbt** in the current working directory.
-dupecheck	(None)	No	Enables duplicate file detection. If two or more images are pixel-for-pixel identical, only one copy of the image data is stored in the .xbt file. This is done by calculating an MD5 hash of the pixel data. Highly recommended for reducing final file size.
-info <file>	File Path	**Yes** (for info)	Puts the tool in "Information Display" mode. Instead of packing, it reads the specified .xbt file and prints its contents. This flag overrides all packing-related flags.
-verbose	(None)	No	Enables verbose output from the image decoders, printing which decoder is being used for each file.

---

3. Input Specifications

3.1. Directory Structure

When packing, you provide a single input directory. The tool traverses it recursively. The relative path of each image from the input directory is preserved inside the .xbt file.

- **Example:** If -input is C:\MySkin\media and it finds an image at C:\MySkin\media\buttons\home.png, the internal path stored in the .xbt file will be buttons/home.png.
- **Note:** Directory separators are normalized to forward slashes (/).

3.2. Supported Image Formats

The tool can decode and process the following formats:

- **PNG (Preferred):** Best for UI elements due to its lossless quality and alpha channel (transparency) support.
- **JPG:** Suitable for large background images where lossy compression is acceptable.
- **GIF:** Supported for both static and animated images. Each frame of an animated GIF will be stored as a frame in the XBT texture.

3.3. File Path Handling

The tool is built to handle Unicode file paths correctly, which is critical on Windows.

- On **Windows**, it uses the CommandLineToArgvW API to get arguments as UTF-16 and converts them internally to UTF-8 for processing. This means file paths with non-ASCII characters will work correctly.
- On **other platforms** (Linux, macOS), it assumes standard UTF-8 command-line arguments.

---

4. Output Formats

This is the most critical section for integration with your Python application, as you will be parsing the console output.

4.1. The .xbt File

This is the primary output of the packing process. It is a binary file containing:

- A header with magic number (XBTF), version (3), and a table of contents.
- The table of contents lists every file path, its dimensions, format, and pointers to its image data.
- The compressed (or uncompressed) pixel data for all unique images.

4.2. Console Output (Packing Mode)

When running in packing mode, the tool prints progress information to stdout. Your application can capture this stream to display progress.

The format is as follows:

1. **Initial Message:**
   PROGRESS:0:Starting to pack [N] textures...
   - [N] is the total number of image files found.
2. **Per-File Processing (repeated for each file):**

   a. **File Path:** The relative path of the file being processed.
   [relative/path/to/image.png]

   b. **Duplicate Message (optional):** If -dupecheck is on and a duplicate is found.
   **** duplicate of [original/path/to/image.png]

   c. **Frame Details (for new or updated files):** A detailed breakdown for each frame in the image. For static images, this appears once. For animated GIFs, it repeats for each frame.
   frame [i] (delay:[ms]) [FMT][A] ([W],[H] @ [S] bytes)

   - [i]: The frame index (starts at 0).
   - [ms]: The frame duration in milliseconds (for animations).
   - [FMT]: The internal texture format after optimization (e.g., RGBA8, BGRA8, R8 , RG8).
   - [A]: An asterisk (*) if the image has no alpha channel (is fully opaque), otherwise a space.
   - [W]: Image width in pixels.
   - [H]: Image height in pixels.
   - [S]: The original, uncompressed size of the frame data in bytes.

   d. **Progress Update:** A machine-readable progress update.
   PROGRESS:[P]:Processed [relative/path/to/image.png]

    - [P] is the completion percentage (0-100).
3. **Finalization Messages:**
   PROGRESS:100:Finalizing XBT file header...
   PROGRESS:100:Packing complete.

4.3. Console Output (-info Mode)

This is the format you requested specifically. It is designed to be human-readable.

1. **Header:**
   Reading info from: [path/to/your/file.xbt]
   ----
2. **Body (repeated for each texture):**
   Texture: [internal/path/to/image.png]
    - **If the texture is a static image (1 frame):**
      Dimensions: [width]x[height]
    - **If the texture is an animated image (>1 frame):**
      Frames: [total_frame_count]
      - Frame [index]: [width]x[height] (delay: [duration]ms)
      (The line for each frame is repeated)
3. **Footer:**

   Total textures: [total_texture_count]
   Total images (frames): [total_frame_and_image_count]

---

5. Key Internal Mechanisms & Optimizations

Understanding these helps explain the tool's output.

1. **Image Channel Reduction:** Before packing, the tool analyzes each 32-bit BGRA image to see if it can be stored more efficiently.
    - **Grayscale + Alpha:** If Red, Green, and Blue channels are identical (R=G=B), it converts the image to two channels (Grayscale + Alpha) and stores it as KD_TEX_FMT_SDR_RG8.
    - **Grayscale Opaque:** If R=G=B and Alpha is fully opaque, it converts to a single-channel KD_TEX_FMT_SDR_R8.
    - **Pure Alpha Mask:** If R, G, and B are all pure white, it converts to a single-channel image using only the Alpha channel (KD_TEX_FMT_SDR_R8).
    - This optimization significantly reduces memory usage for common UI assets.
2. **LZO Compression:** By default (FLAGS_USE_LZO), the tool attempts to compress the pixel data of each image using the LZO1x algorithm. If the compressed data is not smaller than the original, it stores the data uncompressed to avoid wasting space.
3. **Duplicate Detection (-dupecheck):** When enabled, it calculates an MD5 hash of the raw, optimized pixel data for each image. If an identical hash has already been processed, it simply points the new texture entry to the existing image data block instead of writing the data again.

---

6. Practical Workflow Example

Scenario: Pack the contents of C:\SkinProject\media into C:\Output\MySkin.xbt.

Directory Structure:

Generated code

```
    C:\SkinProject\media\
├── background.jpg
├── buttons\
│   ├── button-focus.png
│   └── button-unfocus.png
└── busy-animations\
    └── loading.gif
```

(Assume button-unfocus.png is a pixel-for-pixel copy of button-focus.png)

Command:

Generated batch

```
    TexturePacker.exe -input C:\SkinProject\media -output C:\Output\MySkin.xbt -dupecheck
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Batch
IGNORE_WHEN_COPYING_END

Expected Console Output (Packing):

Generated code

```
    PROGRESS:0:Starting to pack 4 textures...
background.jpg
  frame    0 (delay:   0)                    BGRA8* (1920,1080 @ 8294400 bytes)
PROGRESS:25:Processed background.jpg
buttons/button-focus.png
```

```
   frame    0 (delay:   0)                    BGRA8  (256,64 @ 65536 bytes)
PROGRESS:50:Processed buttons/button-focus.png
buttons/button-unfocus.png
****  duplicate of buttons/button-focus.png
PROGRESS:75:Processed buttons/button-unfocus.png
busy-animations/loading.gif
   frame    0 (delay:  80)                    BGRA8  (32,32 @ 4096 bytes)
   frame    1 (delay:  80)                    BGRA8  (32,32 @ 4096 bytes)
   ... (more frames) ...
PROGRESS:100:Processed busy-animations/loading.gif
PROGRESS:100:Finalizing XBT file header...
PROGRESS:100:Packing complete.
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

Command to inspect the result:

Generated batch

```
TexturePacker.exe -info C:\Output\MySkin.xbt
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Batch
IGNORE_WHEN_COPYING_END

Expected Console Output (-info):

Generated code

```
    Reading info from: C:\Output\MySkin.xbt
----------------------------------------
Texture: background.jpg
  Dimensions: 1920x1080
Texture: buttons/button-focus.png
  Dimensions: 256x64
Texture: buttons/button-unfocus.png
  Dimensions: 256x64
Texture: busy-animations/loading.gif
  Frames: 8
    - Frame 0: 32x32 (delay: 80ms)
    - Frame 1: 32x32 (delay: 80ms)
    ... (etc) ...
----------------------------------------
Total textures: 4
Total images (frames): 11
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

XBTF Extractor Tool Primer

1. Overview

Purpose: xbtfextractor is a command-line utility designed to decompile, or "unpack," .xbt texture archives. It reads a binary .xbt file, processes its contents, and extracts the individual images into a standard directory structure on the filesystem.

Primary Benefit: This tool is the inverse of TexturePacker. It allows developers, skinners, and artists to inspect the contents of a compiled texture archive, modify individual assets, and debug texture-related issues. It correctly handles various internal storage formats and re-encodes them into common image formats like PNG, JPG, and GIF.

Core Functions:

1. **Extraction:** Reads a .xbt file and extracts all contained images to a specified output directory, optionally recreating the original directory structure.
2. **Selective Extraction:** Extracts a single, specific file from the archive by its internal path.
3. **Listing:** Prints a list of all file paths contained within an archive without extracting them.

---

2. Command-Line Usage

The tool's operation is controlled by command-line flags. The final non-option argument is always treated as the input .xbt file.

Flag(s)	Argument	Required?	Description
-h, --help	(None)	No	Displays the built-in usage information and exits.

-p, --print	(None)	No	**List Mode:** Prints all internal file paths from the .xbt archive to the console, one per line. No files are extracted.
-o <dir>, --outdir=<dir>	Directory Path	No	Specifies the root directory where extracted files will be saved. **If omitted, defaults to the current directory (.)**. Required if using -c.
-c, --create	(None)	No	**Recreate Directory Tree:** When extracting, this flag instructs the tool to recreate the full internal directory structure of the archive within the output directory. If omitted, all files are extracted flatly into the output directory.
-f <path>, --file=<path>	Internal File Path	No	**Single File Mode:** Extracts only the specified file from the archive. The path must match the internal path stored in the .xbt file (case-sensitive). Use -p to find the exact path.

Important Note: The source code contains an older, undocumented long option --create-dirs which is an alias for -c.

---

3. Input and Output

3.1. Input File

The tool takes a single .xbt file as its primary input. It reads the file header to verify the XBTF magic number and then parses the table of contents to understand the archive's structure.

3.2. Output Files

The tool re-encodes the internal texture data into standard image formats based on the original file extension stored in the .xbt archive's metadata.

- If the original file was a .png, it saves a **PNG**.
- If the original file was a .jpg or .jpeg, it saves a **JPG** (with quality 95).

- If the original file was a .gif, it saves a **GIF**.
- If the original file had no extension, it **defaults to saving as PNG**.

3.3. File Path Handling

The extractor is designed to work with Unicode (UTF-8) file paths, which is critical for compatibility with modern operating systems, especially Windows. When creating directories or saving files, it converts the internal UTF-8 paths to the wide-character format required by Windows APIs.

---

4. Console Output Formats

Your application can parse the tool's stdout and stderr streams to get status updates and error information.

4.1. Extraction Mode (stdout)

When extracting files (extractAllFiles or extractFile), the tool provides two types of output to stdout: human-readable status and machine-readable progress.

1. **Initial Message (for extractAllFiles):**
   PROGRESS:0:Starting extraction...
2. **Per-File Processing (repeated for each file):**

   a. **Human-Readable Status:**
   Extracting: [internal/path/to/image.png]

   - This line is printed for every file as it begins processing.

b. **Machine-Readable Progress (for extractAllFiles only):**
PROGRESS:[P]:[filename.ext]

- [P] is the completion percentage (0-100).
- [filename.ext] is the base filename (not the full path) of the file just processed.

3. **Finalization Message:**
PROGRESS:100:Extraction complete.
DEBUG: xbtfextractor finished successfully.

4.2. List Mode (-p / --print)

When using the -p flag, the tool prints each internal file path on a new line. The output is clean and simple, designed for easy parsing or piping.

Format:
File:'[internal/path/to/image1.png]'
File:'[internal/path/to/image2.jpg]'
...

4.3. Error Messages (stderr)

The tool reports all warnings and errors to the stderr stream. Your application should monitor this stream to detect failures.

Common Errors:

- ERROR: Cannot open file '[...]': The input .xbt file could not be found or read.
- ERROR: Unusually high number of files (...): A safety check to prevent parsing corrupt files with an impossibly large file count.

- ERROR: Skipping '[...]' due to unreasonable metadata: A safety check against corrupt entries with huge dimensions or file sizes.
- ERROR: Memory allocation failed for [...]: The tool ran out of memory.
- ERROR: --outdir (-o) must be specified when using --create-dirs (-c): A command-line usage error.
- ERROR: No XBT file specified.: A command-line usage error.

---

5. Key Internal Mechanisms

Understanding these helps in troubleshooting and predicting behavior.

1. **File Parsing:** The extractor reads the entire table of contents into memory first. For the extractAllFiles function, it then creates a temporary sorted list of these files to ensure a deterministic, alphabetical extraction order.
2. **Data Decompression:** For each texture frame, it performs these steps:
   a. Seeks to the file offset specified in the metadata.
   b. Reads the packed number of bytes.
   c. If packed size is different from unpacked size, it uses the **LZO1x** algorithm to decompress the data.
   d. If the data is stored in a DXT format (DXT1, DXT3, DXT5), it uses the **libsquish** library to decompress the DXT data into a standard 32-bit RGBA pixel buffer.
3. **Image Re-encoding:** After obtaining the raw pixel data, it re-encodes it into the target image format using well-known libraries:
   - **PNG:** libpng
   - **JPEG:** libjpeg
   - **GIF:** giflib (using its internal quantization algorithm to generate a 256-color palette).
4. **Path Creation (-c):** When the -c flag is used, it parses the destination file path, finds the last directory separator (\), and recursively creates the necessary parent directories before saving the file. It maintains an internal cache of created paths to avoid redundant system calls.

6. Practical Workflow Examples

Scenario: An archive named MySkin.xbt exists in the current directory.

1. List all contents:

Generated batch

```
xbtfextractor.exe -p MySkin.xbt
```

Expected Console Output (stdout):

Generated code

```
File:'background.jpg'
File:'buttons/button-focus.png'
File:'buttons/button-unfocus.png'
File:'busy-animations/loading.gif'
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

2. Extract all files into a new Extracted_Files directory, preserving structure:

Generated batch

```
xbtfextractor.exe -c -o Extracted_Files MySkin.xbt
```

IGNORE_WHEN_COPYING_START

content_copy download
Use code with caution. Batch
IGNORE_WHEN_COPYING_END

Expected Console Output (stdout):

Generated code

```
    PROGRESS:0:Starting extraction...
Extracting: background.jpg
PROGRESS:25:background.jpg
Extracting: buttons/button-focus.png
PROGRESS:50:button-focus.png
Extracting: buttons/button-unfocus.png
PROGRESS:75:button-unfocus.png
Extracting: busy-animations/loading.gif
PROGRESS:100:loading.gif
PROGRESS:100:Extraction complete.
DEBUG: xbtfextractor finished successfully.
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

Resulting Directory Structure:

Generated code

```
    Extracted_Files/
├── background.jpg
├── buttons/
│   ├── button-focus.png
│   └── button-unfocus.png
└── busy-animations/
    └── loading.gif
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

3. Extract just the loading animation into the current directory:

Generated batch

```
    xbtfextractor.exe -f busy-animations/loading.gif MySkin.xbt
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Batch
IGNORE_WHEN_COPYING_END

Expected Console Output (stdout):

Generated code

```
    Extracting: busy-animations/loading.gif
DEBUG: xbtfextractor finished successfully.
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

Resulting File: A single loading.gif file in the current directory.