Search Free

arduino

Posts Wiki

Posted by u/frankcohen just now

Using TAR files in ESP32 with SD applications for easy multiple file transfers

I am engineering a watch to play home videos of my children on my wrist - my Reflections project. I need an easy way to move multiple files from a Cloud based service to the watch. tar and GZip are widely used standards to build and compress an archive of binary (for example, movies and sound files) and text (for example, JSON encoded procedural scripting command) files. Using tar I simplify my code to move one tar file instead of multiple individual files. The ESP32-targz library combines uzlib and TinyUntar to decompress and inflate tar files and works well with Arduino IDE 1.8.13, ESP32, and an SD card. This is my developer's journal of the problems that I encountered and the work-arounds I used.

I love the ESP32-targz library. It is elegantly written using callbacks to uzlib and TinyUtar. Error and debug reporting is strong. It installs as a standard Zip file into the Arduino IDE Libraries. I used Atom editor to look into and search the source code. And it compiles easily using Arduino IDE as a C and C++ library. I spent around 30 hours getting it to work with my ESP32 and SD card set-up. I bumped into a bunch of problems of my own making, and some that the library just didn't handle.

Using tar files in a communication sketch delivers these benefits:

1. Archives move multiple files at once, less chance for errors during transmission and easier coding
2. Archives guarantee their file contents are the same after transfer
3. Archives decompress into a file/directory structure on the SD card

Tar is not perfect for ESP32 and embedded systems. Zip decompression libraries require lots of memory and are more complicated to code. Tar standard supports many extensions that are not supported on ESP32 SD systems - for example, extended file attributes from MacOS archives - and this makes code larger and more complicated.

ESP32-targz library implements the minimum needed tar and gzip features to inflate (uncompress) tar files.

Library requires this to be defined ahead of the include to work with SD cards. It only works this way:

```
#define DEST_FS_USES_SD
#include <ESP32-targz.h>
```

Define it in any other order and ESP32-targz saves inflates files to the ESP32 SPIFFS Filesystem in the flash memory.

```
  bool ret = false;
  const char* tarGzFile = "/startup.tar.gz";
  const char * mytmpdest = "/tempo";

  TarGzUnpacker *TARGZUnpacker = new TarGzUnpacker();

  TARGZUnpacker->haltOnError( true ); // stop on fail (manual restart/reset required)
  TARGZUnpacker->setTarVerify( true ); // true = enables health checks but slows down

  TARGZUnpacker->setTarStatusProgressCallback( BaseUnpacker::defaultTarStatusProgress
  TARGZUnpacker->setupFSCallbacks( targzTotalBytesFn, targzFreeBytesFn ); // prevent
  TARGZUnpacker->setGzProgressCallback( BaseUnpacker::defaultProgressCallback ); // t
  TARGZUnpacker->setLoggerCallback( BaseUnpacker::targzPrintLoggerCallback  );    //
  TARGZUnpacker->setTarProgressCallback( BaseUnpacker::defaultProgressCallback ); //
  TARGZUnpacker->setTarMessageCallback( myTarMessageCallback ); // tar log verbosity

  Serial.println("Testing tarGzExpander without intermediate file");
  if( !TARGZUnpacker->tarGzExpander(tarGzFS, tarGzFile, tarGzFS, mytmpdest, nullptr )
    Serial.print("tarGzExpander+intermediate file failed with return code ");
    Serial.println( TARGZUnpacker->tarGzGetError() );
  } else {
    ret = true;
  }
```

and this logging callback:

```
char tmp_path[255] = {0};

void myTarMessageCallback(const char* format, ...)
{
  va_list args;
  va_start(args, format);
  vsnprintf(tmp_path, 255, format, args);
  va_end(args);
}
```

Running this code gives me this output:

```
[TAR] startup/Tron.m4a                                                    80656 bytes
Progress: 0% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 25% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 50% ▒▒▒▒▒▒▒▒▒▒▒▒▒
[TAR] startup/Candle.mpg                                                 766031 bytes
Progress: 0% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 25% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 50% ▒▒▒▒▒▒▒▒▒▒▒▒▒
[TAR] startup/Hurry.mpg                                                 1845676 bytes
Progress: 0% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 25% ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ 50% ▒▒▒▒▒▒▒▒▒▒▒▒▒
```

I made a bunch of mistakes:

directory names.

There were warnings in the ESP32-targz document "tarGzExpander: files smaller than 4K aren't processed." This turns out to be wrong. The library handles smaller files normally.

File extensions confuse the library. For example, I included startup.ref. Library stopped decompressing on startup.ref with no error. Solved by using .txt file extension.

MacOS files confuse the library. MacOS comes with bsdtar installed, and symlink-ed to tar. The library stops decompressing at ./DS_Store, .Spotlight-V100 with no error. The solution is to use the undocumented flag COPYFILE_DISABLED when creating the tar and exclude the Mac specific files from the archive:

```
#! /bin/bash
cd /Users/frankcohen/Desktop/ReflectionsExperiments
rm startup.tar.gz
COPYFILE_DISABLE=1 tar czf startup.tar.gz --exclude ".DS_Store" startup
```

Unfortunately bsdtar adds text files to archives using extended file attributes (T_EXTENDED) and ESP32-targz ignores these files, and stops inflating an archive with an error message:

```
[D][ESP32-targz-lib.cpp:574] tarHeaderCallBack(): Ignoring extended data.
[E][ESP32-targz-lib.cpp:784] tarStreamWriteCallback(): [TAR ERROR] Written length dif
```

The work around is to modify ESP32-targz:

```
line 45 of https://github.com/tobozo/ESP32-targz/blob/master/src/ESP32-targz-lib.cpp
changed to: if( ( proper->type == TAR::T_NORMAL ) || ( proper->type == TAR::T_EXTENDE

and line 652 of line 45 of https://github.com/tobozo/ESP32-targz/blob/master/src/ESP3
if( ( proper->type == TAR::T_NORMAL ) || ( proper->type == TAR::T_EXTENDED ) ) {
```

I posted the above patch as Issue 33 to the ESP32-targz project in April 2021.

Without these changes to detect TAR:: T_EXTENDED() an 11,054 uncompressed file containing JSON encoded data ends the TAR decompress feature, regardless of more files in the tar archive to inflate/decompress.

ESP32-targz comes with a lot of logging; However, most of the logging is turned-off or coded with special checks to only log on ESP2866 processors. I had to add this code to line 256 of ESP32-targz-lib.h to get debug and informational logging to appear in the Serial Monitor:

```
#define log_n(format, ...) printf(ARDUHAL_LOG_FORMAT(N, format), ##__VA_ARGS__);
#define log_e(format, ...) printf(ARDUHAL_LOG_FORMAT(E, format), ##__VA_ARGS__);
#define log_w(format, ...) printf(ARDUHAL_LOG_FORMAT(W, format), ##__VA_ARGS__);
#define log_i(format, ...) printf(ARDUHAL_LOG_FORMAT(I, format), ##__VA_ARGS__);
```

```
[TGZ] Will expand without intermediate file
[TGZ] Will direct-expand /startup.tar.gz to /tempo
[I][ESP32-targz-lib.cpp:989] gzReadHeader(): [GZ INFO] valid gzip file detected! gz s
[I][ESP32-targz-lib.cpp:998] gzReadHeader(): [GZ INFO] Available space:-1399848960 by
[TAR DEBUG]: entering tar setup
```

Conclusion

In my experience ESP32-targz is excellent: one library including TinyUntar and uzlib, Arduino IDE compatibility, and fairly rich logging. I had my own misunderstandings (above) that kept me from being successful.

My watch project is called Reflections and is hosted here.