Customizing Closed Source Trail Camera Firmware: Toolset and Case Study

Robert Zak (Robert(dot)zak(at)comcast(dot)net

# Abstract

Current commercial trail camera models are very good. In general, they are inexpensive, readily available, easy to deploy and recover, energy efficient, feature rich, and with increasingly high video and photo quality. None-the-less, they are not an ideal fit for all applications. In these cases, the alternatives are either to use existing trail cameras and work around their limitations, or to build an entire new "home brew" camera.

This work focuses on a third alternative: customizing the camera firmware to add application-specific functionality to a standard, commercial trail camera. Because the firmware in commercial trail cameras is typically "closed source" and available only in binary format, this solution includes a software toolchain which can analyze the firmware, and allow the insertion of custom code, written in a high level language, with full access to functions and data found in the "factory" install. This customization can be focused on application-specific features without requiring modification most of the system firmware, and complete re-use of the trail camera "physical plant".

Adding code based on this tool chain is a much lower effort to than writing an entire new trail camera application. It is analogous to developing and “App” environment for Trail Cameras similar to that of smart phones, tablets, and computers.

This software toolchain can also be used to support physical modifications of the commercial trail camera.

In an example, I cover the conversion of a standard trail camera (Browning BTC-7A) from IR to white flash, the addition of a readable date/time in the "playback" mode, the addition of a stateful battery capacity monitor, and over-riding internal restrictions on the length of LED-illuminated videos.

# Overview of Toolchain:

An overview of the toolchain is shown in figure XX.

The goal of the tool chain is to produce a firmware image, containing a consistent combination of factory firmware, and application-specific user defined code. This image can be installed into the trail camera using the firmware upgrade mechanism.

Key parts of the tool chain include:

* A list of available functions (or “services”) provided by the base camera software environment. These can be called by custom apps to access the camera’s file system, sensors, settings, and display.
* A list of camera events which can be used as entry points for new code.
* Cross compiler environment for building custom functions
* C-Support: This toolchain uses cross compilers, linkers, and loaders from the gcc suite. The target depends on the trail camera. In the case of the BTC-7A, the target is the MIPS architecture (32-bit, Little Endian).
* Tool for reassembling .BRN, creating an image with augmented functionality.

The tool chain is designed to run primarily in free cloud compute resources (Google Colab).

Application-specific functions are written in C, and can be put into different files for modularization.

To make room for the application-specific code, the tool overwrites some functions in the factory firmware base which are not required for normal camera operation. The tool maintains a list of such functions to hijack, and automatically chooses enough to provide the space in the binary for new code. Currently, this list of functions makes available some >100KB of memory – large enough for fairly complex customized functions.

Global Variables:

Patch List:

Combining Tools:

Baseline Firmware Image (BTC.BRN)

In addition to Ghidra, there are a number of python-based functions for

Examples:

Here are several examples, of increasingly complexity, of code to customize at BTC-7A

Example 1: Large Print Date/Time on Photo/Video Review

When reviewing photos in the field, it is often valuable to know the date and time for the photo or video being reviewed. This information is available in the “info strip” at the bottom of the photo or video. Unfortunately, the strip is very difficult to read on the 2” LCD screen. This patch addresses this problem by adding the date and time, in larger font, to the bottom of the review screen.

Example 2: Stateful Battery Monitor

When using a popular type of battery based on LiFeS2 (e.g. Energizer Ultimate Lithium) batteries, the factor battery monitor, which attempts to map current battery pack voltage to remaining capacity, is of little use. In fact, due to the battery chemistry and unique construction of these batteries, the manufacturer likewise notes <<Reference>> that a loaded test, or pulsed test, or full AC series resistance test are also poor indicators of the state of charge of batteries. The manufacturer gives no example of any single measurement which does reliably reflect the state of charge of this type of battery.

An alternative is to use a stateful battery monitor. Basically, to keep track of the energy consumed by the camera while in standby, active photo/video mode, or during usage of the User Interface, and to present this as a fraction of the total energy available in a new battery pack. The battery monitor app implements this function for LiFeS2 and alkaline batteries, respectively, including adjustments for temperature and rate of discharge per manufacturer specifications.

Reference: "Open source trail cameras"

Reference: Energizer Ultimate Lithium Batteres