UAB's V2F compression system

1.0

Generated by Doxygen 1.8.17

Chapter 1

Requirements

Global main (void)

V2F-3.1

Global test_compression_decompression_minimal_codec (void)

V2F-1.1, V2F-1.2, V2F-1.3, V2F-1.4

Global test_compression_decompression_steps (void)

V2F-1.1, V2F-1.2, V2F-1.3, V2F-1.4, V2F-2.1

Global test_compressor_create (void)

V2F-1.3

Global test_decorrelator_create (void)

V2F-1.1

Global test_decorrelator_prediction_mapping (void)

V2F-1.1.2

Global test_quantizer_create (void)

V2F-2.1

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

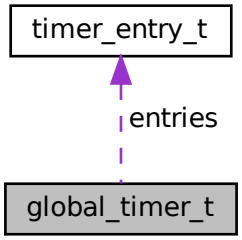
Chapter 4

Data Structure Documentation

4.1 global_timer_t Struct Reference

`#include <timer.h>`

Collaboration diagram for global_timer_t:



Data Fields

- timer_entry_t entries [256]
- uint16_t entry_count

4.1.1 Detailed Description

Struct holding timer entries.

4.1.2 Field Documentation

4.1.2.1 entries

```
timer_entry_t global_timer_t::entries[256]
```

Timer entries.

4.1.2.2 entry_count

```
uint16_t global_timer_t::entry_count
```

Number of used entries.

The documentation for this struct was generated from the following file:

- timer.h

4.2 timer_entry_t Struct Reference

```
#include <timer.h>
```

Data Fields

- char name [256]

Current run values (for the last start/stop cycle)

- bool running
- clock_t clock_before
- clock_t clock_after
- double wall_before
- double wall_after

Global run tracking

- uint64_t count
- double total_cpu_s
- double total_wall_s

4.2.1 Detailed Description

Structure representing one timer entry.

4.2.2 Field Documentation

4.2.2.1 clock_after

```
clock_t timer_entry_t::clock_after
```

Clock value when the timer was stopped (if it was stoppeD).

4.2.2.2 clock_before

```
clock_t timer_entry_t::clock_before
```

Clock value when the timer was started.

4.2.2.3 count

```
uint64_t timer_entry_t::count
```

Number of start/stop cycles a timer has been run.

4.2.2.4 name

```
char timer_entry_t::name[256]
```

Timer name.

4.2.2.5 running

```
bool timer_entry_t::running
```

Is the timer running?

4.2.2.6 total_cpu_s

```
double timer_entry_t::total_cpu_s
```

Total CPU time in seconds for all start/stop cycles.

4.2.2.7 total_wall_s

```
double timer_entry_t::total_wall_s
```

Total wall time in seconds for all start/stop cycles.

4.2.2.8 wall_after

```
double timer_entry_t::wall_after
```

Wall time value when the timer was stopped (if it was stoppeD).

4.2.2.9 wall_before

```
double timer_entry_t::wall_before
```

Wall time value when the timer was started.

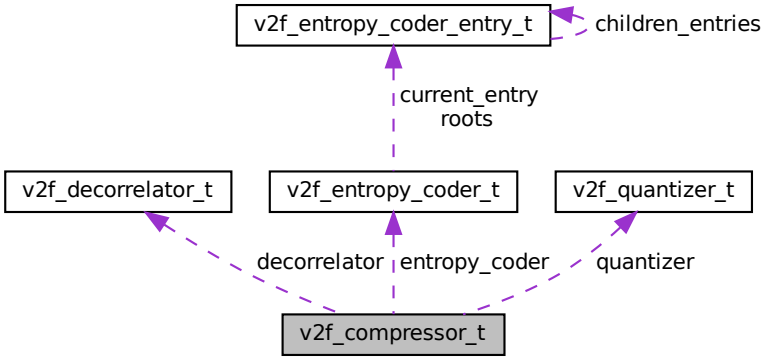
The documentation for this struct was generated from the following file:

- timer.h

4.3 v2f_compressor_t Struct Reference

```
#include <v2f.h>
```

Collaboration diagram for v2f_compressor_t:



Data Fields

- v2f_quantizer_t * quantizer
- v2f_decorrelator_t * decorrelator
- v2f_entropy_coder_t * entropy_coder

4.3.1 Detailed Description

Represent a complete compression pipeline.

4.3.2 Field Documentation

4.3.2.1 decorrelator

```
v2f_decorrelator_t* v2f_compressor_t::decorrelator
```

Pointer to the decorrelator to be used.

4.3.2.2 entropy_coder

```
v2f_entropy_coder_t* v2f_compressor_t::entropy_coder
```

Pointer to the entropy coder to be used.

4.3.2.3 quantizer

```
v2f_quantizer_t* v2f_compressor_t::quantizer
```

Pointer to the quantizer to be used.

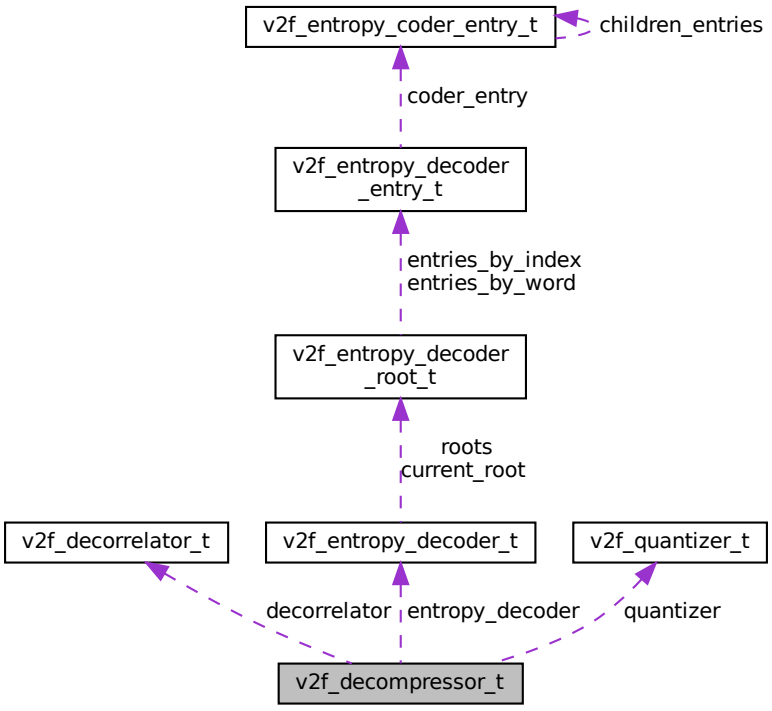
The documentation for this struct was generated from the following file:

- v2f.h

4.4 v2f_decompressor_t Struct Reference

`#include <v2f.h>`

Collaboration diagram for v2f_decompressor_t:



Data Fields

- v2f_quantizer_t * quantizer
- v2f_decorrelator_t * decorrelator
- v2f_entropy_decoder_t * entropy_decoder

4.4.1 Detailed Description

Represent a complete compression pipeline.

4.4.2 Field Documentation

4.4.2.1 decorrelator

```
v2f_decorrelator_t* v2f_decompressor_t::decorrelator
```

Pointer to the decorrelator to be used.

4.4.2.2 entropy_decoder

```
v2f_entropy_decoder_t* v2f_decompressor_t::entropy_decoder
```

Pointer to the entropy decoder to be used.

4.4.2.3 quantizer

```
v2f_quantizer_t* v2f_decompressor_t::quantizer
```

Pointer to the quantizer to be used.

The documentation for this struct was generated from the following file:

- v2f.h

4.5 v2f_decorrelator_t Struct Reference

```
#include <v2f.h>
```

Data Fields

- v2f_decorrelator_mode_t mode
- v2f_sample_t max_sample_value
- uint64_t samples_per_row

4.5.1 Detailed Description

Represent a decorrelator, one of the stages of the compression pipeline.

4.5.2 Field Documentation

4.5.2.1 max_sample_value

```
v2f_sample_t v2f_decorrelator_t::max_sample_value
```

Maximum original sample value.

4.5.2.2 mode

```
v2f_decorrelator_mode_t v2f_decorrelator_t::mode
```

Decorrelation mode.

4.5.2.3 samples_per_row

```
uint64_t v2f_decorrelator_t::samples_per_row
```

Samples per row, aka stride. If 0 or equal to the number of samples, the input will be processed as a single row matrix. This is the default. Prediction modes that require 2D geometry information can use this value to perform their computations.

The documentation for this struct was generated from the following file:

- v2f.h

4.6 v2f_entropy_coder_entry_t Struct Reference

```
#include <v2f.h>
```

Collaboration diagram for v2f_entropy_coder_entry_t:

Data Fields

- struct v2f_entropy_coder_entry_t ** children_entries
- uint32_t children_count
- uint8_t * word_bytes

4.6.1 Detailed Description

Each of the individual table entries, corresponding to an included element in a V2F tree.

4.6.2 Field Documentation

4.6.2.1 children_count

`uint32_t v2f_entropy_coder_entry_t::children_count`

Number of children instances.

4.6.2.2 children_entries

`struct v2f_entropy_coder_entry_t** v2f_entropy_coder_entry_t::children_entries`

List of `children_count` pointers to other v2f_entropy_coder_entry_t instances.

4.6.2.3 word_bytes

`uint8_t* v2f_entropy_coder_entry_t::word_bytes`

Pointer to a buffer with the bytes corresponding to this entry's word bytes. Emitting this entry consists in outputing these bytes.

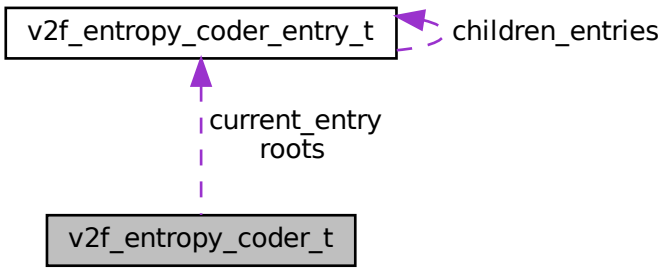
The documentation for this struct was generated from the following file:

- v2f.h

4.7 v2f_entropy_coder_t Struct Reference

`#include <v2f.h>`

Collaboration diagram for v2f_entropy_coder_t:



Data Fields

General coder parameters

- uint8_t bytes_per_word
- v2f_sample_t max_expected_value

Forest structure

- v2f_entropy_coder_entry_t ** roots
- uint32_t root_count

Auxiliary for efficient coding

- v2f_entropy_coder_entry_t * current_entry

4.7.1 Detailed Description

Represent a generic variable to fixed (V2F) coder

4.7.2 Field Documentation

4.7.2.1 bytes_per_word

```
uint8_t v2f_entropy_coder_t::bytes_per_word
```

Number of bytes used to represent the word of an included treee node.

4.7.2.2 current_entry

```
v2f_entropy_coder_entry_t* v2f_entropy_coder_t::current_entry
```

Current node in the V2F forest.

4.7.2.3 max_expected_value

```
v2f_sample_t v2f_entropy_coder_t::max_expected_value
```

Maximum sample value expected by this coder.

4.7.2.4 root_count

```
uint32_t v2f_entropy_coder_t::root_count
```

Number of root entries in the coder.

4.7.2.5 roots

```
v2f_entropy_coder_entry_t** v2f_entropy_coder_t::roots
```

List of root entries for this coder (don't count towards `entry_cont`).

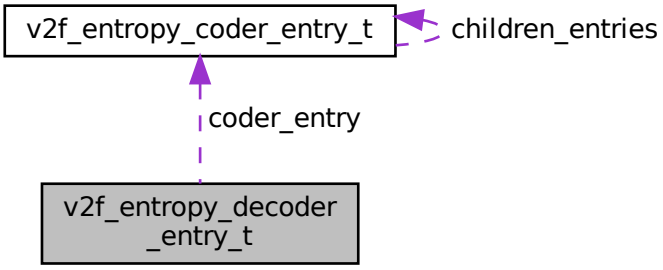
The documentation for this struct was generated from the following file:

- v2f.h

4.8 v2f_entropy_decoder_entry_t Struct Reference

```
#include <v2f.h>
```

Collaboration diagram for v2f_entropy_decoder_entry_t:



Data Fields

- v2f_sample_t * samples
- uint32_t sample_count
- uint32_t children_count
- v2f_entropy_coder_entry_t * coder_entry

4.8.1 Detailed Description

Each of the entries within a V2F tree.

4.8.2 Field Documentation

4.8.2.1 children_count

```
uint32_t v2f_entropy_decoder_entry_t::children_count
```

Number of children of this entry.

4.8.2.2 coder_entry

```
v2f_entropy_coder_entry_t* v2f_entropy_decoder_entry_t::coder_entry
```

Pointer to the twin entry in the corresponding coder. (Not used during decompression, only when dumping the coder/decoder pair with v2f_file.h

4.8.2.3 sample_count

```
uint32_t v2f_entropy_decoder_entry_t::sample_count
```

Number of samples associated to this entry.

4.8.2.4 samples

```
v2f_sample_t* v2f_entropy_decoder_entry_t::samples
```

Samples associated to this entry.

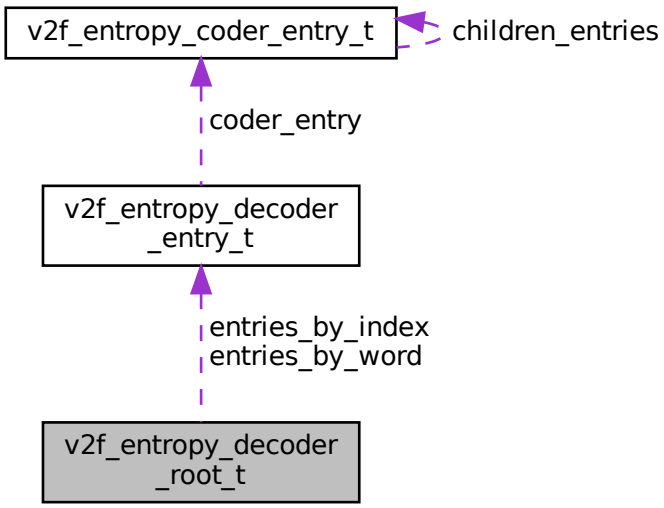
The documentation for this struct was generated from the following file:

- v2f.h

4.9 v2f_entropy_decoder_root_t Struct Reference

`#include <v2f.h>`

Collaboration diagram for v2f_entropy_decoder_root_t:



Data Fields

- v2f_entropy_decoder_entry_t * entries_by_index
- uint32_t root_entry_count
- v2f_entropy_decoder_entry_t ** entries_by_word
- uint32_t root_included_count

4.9.1 Detailed Description

This type gives access to all entries within a V2F tree.

4.9.2 Field Documentation

4.9.2.1 entries_by_index

`v2f_entropy_decoder_entry_t* v2f_entropy_decoder_root_t::entries_by_index`

Array of entries ordered by index value.

4.9.2.2 entries_by_word

```
v2f_entropy_decoder_entry_t** v2f_entropy_decoder_root_t::entries_by_word
```

Array of pointers to included nodes, so that they can be properly decoded.

4.9.2.3 root_entry_count

```
uint32_t v2f_entropy_decoder_root_t::root_entry_count
```

Total number of entries in `entries_by_index`.

4.9.2.4 root_included_count

```
uint32_t v2f_entropy_decoder_root_t::root_included_count
```

Total number of entries in `entries_by_word` which have an associated codeword. These are indexed by the codewords to be read from compressed blocks (unsigned, big endian, `bytes_per_word` bytes).

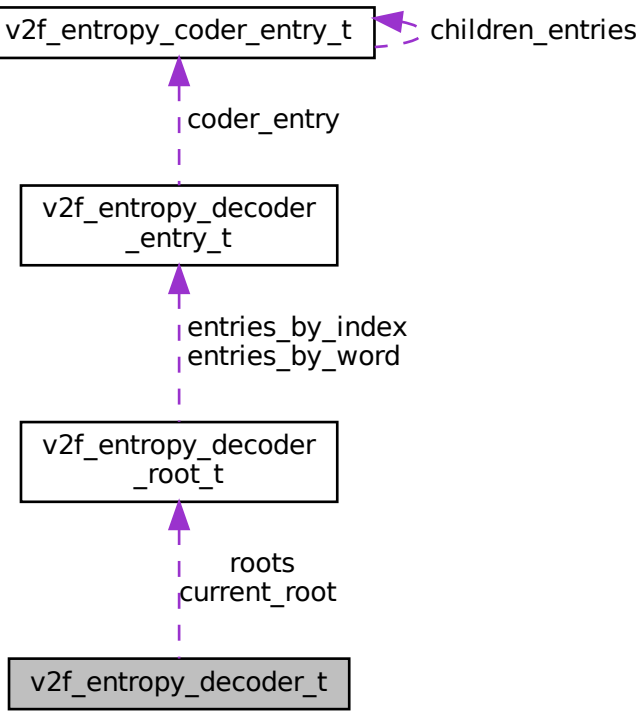
The documentation for this struct was generated from the following file:

- v2f.h

4.10 v2f_entropy_decoder_t Struct Reference

```
#include <v2f.h>
```

Collaboration diagram for v2f_entropy_decoder_t:



Data Fields

- uint8_t bytes_per_word
- uint8_t bytes_per_sample
- v2f_entropy_decoder_root_t ** roots
- uint32_t root_count
- v2f_entropy_decoder_root_t * current_root

4.10.1 Detailed Description

Represent a V2F decoder

4.10.2 Field Documentation

4.10.2.1 bytes_per_sample

```
uint8_t v2f_entropy_decoder_t::bytes_per_sample
```

Number of bytes used to represent each original sample value.

4.10.2.2 bytes_per_word

```
uint8_t v2f_entropy_decoder_t::bytes_per_word
```

Number of bytes per index expected in the compressed data.

4.10.2.3 current_root

```
v2f_entropy_decoder_root_t* v2f_entropy_decoder_t::current_root
```

Auxiliary pointer to the current root, useful for efficient decoding.

4.10.2.4 root_count

```
uint32_t v2f_entropy_decoder_t::root_count
```

Number of roots in this decoder.

4.10.2.5 roots

```
v2f_entropy_decoder_root_t** v2f_entropy_decoder_t::roots
```

List of root nodes, indexed by a simple index s.

- Root [s] should be able to code any symbol >= s.
- Roots can be aliased (multiple pointers to the same root)
- There should be at least m elements in this list if m-1 is the maximum expected sample value.

The documentation for this struct was generated from the following file:

- v2f.h

4.11 v2f_quantizer_t Struct Reference

```
#include <v2f.h>
```

Data Fields

- v2f_quantizer_mode_t mode
- v2f_sample_t step_size

4.11.1 Detailed Description

Represent a quantizer of input samples.

4.11.2 Field Documentation

4.11.2.1 mode

`v2f_quantizer_mode_t v2f_quantizer_t::mode`

Index to identify the quantization being applied.

4.11.2.2 step_size

`v2f_sample_t v2f_quantizer_t::step_size`

Maximum number of input sample values per quantization bin. Set to 1 for no quantization.

The documentation for this struct was generated from the following file:

- v2f.h

4.12 v2f_test_sample_t Struct Reference

`#include <test_samples.h>`

Data Fields

- const char ∗const path
- const char ∗const description
- const uint32_t bytes

4.12.1 Detailed Description

Represent all needed information about a test sequence.

4.12.2 Field Documentation

4.12.2.1 bytes

```
const uint32_t v2f_test_sample_t::bytes
```

Number of bytes available in the file.

4.12.2.2 description

```
const char* const v2f_test_sample_t::description
```

Optional information about the sample.

4.12.2.3 path

```
const char* const v2f_test_sample_t::path
```

Path to the file with data.

The documentation for this struct was generated from the following file:

- test_samples.h

Chapter 5

File Documentation

5.1 build_test.c File Reference

```
#include <sys/types.h>
#include <stdio.h>
#include <assert.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/v2f_build.h"
#include "../src/v2f_entropy_coder.h"
#include "../fuzzing/fuzzing_common.h"
#include "../src/errors.h"
#include "../src/timer.h"
#include "../src/log.h"
```

Functions

- void test_build_minimal_entropy (void)
- void test_build_minimal_codec (void)
- void register_build (void)

5.1.1 Function Documentation

5.1.1.1 register_build()

```
void register_build (
            void  )
```

Register the build suite

5.1.1.2 test_build_minimal_codec()

```
void test_build_minimal_codec (
            void  )
```

Test the minimal compressor/decompressor pair produced by v2f_build_minimal_codec.

5.1.1.3 test_build_minimal_entropy()

```
void test_build_minimal_entropy (
            void  )
```

Test the minimal entropy coder and decoder produced by v2f_build_minimal_forest.

5.2 command_line_fiu_fuzzer.c File Reference

```
#include <fenv.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>
#include <execinfo.h>
#include <fiu.h>
#include <fiu-control.h>
#include "command_line_fuzzer_common.h"
```

Macros

- #define _GNU_SOURCE
- #define FIU_ENABLE

Functions

- int main (int argc, char *argv[])
- int fiu_callback (const char *name, int *failnum, void **failinfo, unsigned int *flags)
- int call_command (int(*command)(int argc, char *argv[]), int argc, char *argv[], char const *setstdin, bool closestdout, bool closestderr)

5.2.1 Detailed Description

Not an actual fuzzer.

This file takes all fuzzing samples from a directory and processes them. Useful to take measure code coverage for a huge number of samples WHILE INJECTING I/O FAILURES with libfiu.

5.2.2 Macro Definition Documentation

5.2.2.1 _GNU_SOURCE

```
#define _GNU_SOURCE
```

Symbol that allows access to low-level symbols.

5.2.2.2 FIU_ENABLE

```
#define FIU_ENABLE
```

Symbol to enable the injection of I/O failure points.

5.2.3 Function Documentation

5.2.3.1 call_command()

```
int call_command (
            int(*)(int argc, char *argv[]) command,
            int argc,
            char * argv[],
            char const * setstdin,
            bool closestdout,
            bool closestderr )
```

Function that wraps the actual call to the tool.

Parameters

command	command to be called
argc	argc for the command
argv	argv for the command
setstdin	path to the file to be used as *stdin*
closestdout	should stdout be closed?
closestderr	should stderr be closed?

Returns

the return value of the called command

5.2.3.2 fiu_callback()

```
int fiu_callback (
            const char * name,
            int * failnum,
            void ** failinfo,
            unsigned int * flags )
```

Callback for libfiu fiu_enable.

Parameters

name	parameters of the fiu callback function (see libfiu documentation).
failnum	parameters of the fiu callback function (see libfiu documentation).
failinfo	parameters of the fiu callback function (see libfiu documentation).
flags	parameters of the fiu callback function (see libfiu documentation).

Returns

- 1 : the fault was triggered
- 0 : the fault was not triggered

5.2.3.3 main()

```
int main (
            int argc,
            char * argv[] )
```

Take all fuzzing samples from a directory and processes them. Useful to take measure code coverage for a huge number of samples WHILE INJECTING I/O FAILURES with libfiu.

Parameters

argc	command-line argument count
argv	command-line argument list

Returns

0 is always returned by this method (the program can abort, though)

5.3 command_line_fuzzer.c File Reference

```
#include <fenv.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
```

```
#include <string.h>
#include <assert.h>
#include "fuzzing_common.h"
#include "command_line_fuzzer_common.h"
```

Functions

- int call_command (int(∗command)(int argc, char ∗argv[ ]), int argc, char ∗argv[ ], char const ∗setstdin, bool closestdout, bool closestderr)
- int main (int argc, char ∗argv[ ])

5.3.1 Detailed Description

A fuzzing harness that tests command-line programs.

5.3.2 Function Documentation

5.3.2.1 call_command()

```
int call_command (
             int(*)(int argc, char *argv[]) command,
             int argc,
             char * argv[],
             char const * setstdin,
             bool closestdout,
             bool closestderr )
```

This is just a pass-through function so that command_line_fiu_fuzzer can overwrite it.

Parameters

command	command to be tested
argc	main's argc for the command
argv	main's argv for the command
setstdin	file to use as *stdin*
closestdout	should *stdout* be closed?
closestderr	should *stderr* be closed?

Returns

the return value of the call

5.3.2.2 main()

```
int main (
            int argc,
            char * argv[] )
```

A fuzzing harness that tests command-line programs.

Parameters

argc	number of command-line arguments
argv	list of command-line arguments

Returns

the return value of the handled call

5.4 command_line_fuzzer_common.c File Reference

```
#include <unistd.h>
#include <sched.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include <fcntl.h>
#include "command_line_fuzzer_common.h"
#include "command_line_fuzzer_include_helper.h"
```

Macros

- #define fopen fake_fopen
- #define TOOL_NAME verify_codec_bin
- #define TOOL_FILE "../bin/v2f_verify_codec.c"
- #define TOOL_NAME compress
- #define TOOL_FILE "../bin/v2f_compress.c"
- #define TOTAL_TEMP_FILES 256
- #define MAX_COMMANDS 16
- #define MAX_TEMP_FILES 16
- #define FULL_READ(ptr, size, stream)

Enumerations

- enum command_line_element_t {
  CMD_INPLACE_PARAMETER = 0, **CMD_FILE** = 1, **CMD_REUSE_FILE** = 2, **CMD_SETSTDIN** = 3,
  CMD_CLOSESTDOUT = 5, **CMD_CLOSESTDERR** = 6, **CMD_DICT_PARAMETER** = 7, **CMD_DICT_OP↩**
  TION = 8,
  CMD_END_LIST = 9 }

Functions

- FILE * fake_fopen (const char *pathname, const char *mode)
- int fake_open_read (const char *pathname, const pid_t parent_pid)
- static void get_rid_of_temp_files (int how_many)
- static void show_command_line (int tool, int argc, char *argv[ ])
- void handle_command_file (FILE *const file, int tool, char *const scratch_buffer, int(*call_wrapper)(int(*command)(int argc, char *argv[ ]), int argc, char *argv[ ], char const *setstdin, bool closestdout, bool closestderr))

Variables

- static int(* command_list [2])(int argc, char *argv[ ])
- static const char *const temporary_file_names [256]

5.4.1 Detailed Description

Implementation of utility functions used by the command_line_fuzzer.c and command_line_fiu_fuzzer.c fuzzers.

5.4.2 Macro Definition Documentation

5.4.2.1 fopen

```
#define fopen fake_fopen
```

Redefinition of fopen as fake_fopen for safe fuzzer execution

5.4.2.2 FULL_READ

```
#define FULL_READ(
              ptr,
              size,
              stream )
```

Value:

```
if (fread((ptr), (size), 1, (stream)) != 1) { \
    if (ferror(stream)) { abort(); } else { return; } \
}
```

Read *size* bytes from *stream* into *ptr*, or abort.

Parameters

ptr	buffer where read data is to be stored
size	number of bytes to read
stream	FILE* open for reading from which data are to be read

5.4.2.3 MAX_COMMANDS

```
#define MAX_COMMANDS 16
```

Maximum number of commands in an invocation.

5.4.2.4 MAX_TEMP_FILES

```
#define MAX_TEMP_FILES 16
```

Maximum number of temporary files in an invocation.

5.4.2.5 TOOL_FILE [1/2]

```
#define TOOL_FILE "../bin/v2f_verify_codec.c"
```

Name of the file implementing the tool

5.4.2.6 TOOL_FILE [2/2]

```
#define TOOL_FILE "../bin/v2f_compress.c"
```

Name of the file implementing the tool

5.4.2.7 TOOL_NAME [1/2]

```
#define TOOL_NAME verify_codec_bin
```

Name of the tool

5.4.2.8 TOOL_NAME [2/2]

```
#define TOOL_NAME compress
```

Name of the tool

5.4.2.9 TOTAL_TEMP_FILES

```
#define TOTAL_TEMP_FILES 256
```

Total number of available temporary file names

5.4.3 Enumeration Type Documentation

5.4.3.1 command_line_element_t

```
enum command_line_element_t
```

Enumeration of the types of command-line elements available in an invocation

5.4.4 Function Documentation

5.4.4.1 fake_fopen()

```
FILE * fake_fopen (
            const char * pathname,
            const char * mode )
```

Given a (valid) file path to be open and an open mode, open instead a PID-dependent file so that concurrent fuzzer executions do not clash.

Parameters

pathname	path of the file to be used
mode	mode in which the file is to be open

Returns

the fid of the multiprocess-safe file opened in this call

5.4.4.2 fake_open_read()

```
int fake_open_read (
            const char * pathname,
            const pid_t parent_pid )
```

Given a (valid) file path to be open, open instead a PID-dependent file so that concurrent fuzzer executions do not clash.

Parameters

pathname	path of the file to be used
parent_pid	PID of the parent process

Returns

the fid of the multiprocess-safe file opened in this call

5.4.4.3 get_rid_of_temp_files()

```
static void get_rid_of_temp_files (
            int how_many )  [static]
```

Remove PID-dependent temporary files

Parameters

how_many	number of temporary files to remove

Returns

(void)

5.4.4.4 handle_command_file()

```
void handle_command_file (
            FILE *const file,
            int tool,
            char *const scratch_buffer,
            int(*)(int(*command)(int argc, char *argv[]), int argc, char *argv[], char const
*setstdin, bool closestdout, bool closestderr) call_wrapper )
```

Execute and verify a binary/tool on a file.

Parameters

file	file onto which the command is to be tested
tool	command to be evaluated
scratch_buffer	buffer to be used to parse the call
call_wrapper	function that wraps the actual call to the tool

5.4.4.5 show_command_line()

```
static void show_command_line (
            int tool,
            int argc,
            char * argv[] )  [static]
```

Print information about a command-line invocation

Parameters

tool	tool (command) to be invoked
argc	argc of the invocation
argv	argv of the invocation

Returns

(void)

5.4.5 Variable Documentation

5.4.5.1 command_list

```
int(* command_list[2])(int argc, char *argv[])  [static]
```

Initial value:

```
= {
        main_verify_codec_bin,
        main_compress,

}
```

Array of command/tool functions available for command-line testing

5.4.5.2 temporary_file_names

```
const char* const temporary_file_names[256]  [static]
```

Valid temporary file names

5.5 command_line_fuzzer_common.h File Reference

```
#include <stdio.h>
#include <stdbool.h>
```

Macros

- #define SCRATCH_BUFFER_SIZE 1024

Functions

- void handle_command_file (FILE *const file, int tool, char *const scratch_buffer, int(*call_wrapper)(int(*command)(int argc, char *argv[]), int argc, char *argv[], char const *setstdin, bool closestdout, bool closestderr))
- int fake_open_read (const char *pathname, const pid_t parent_pid)

5.5.1 Detailed Description

Declaration of utility functions used by command_line_fuzzer.c and command_line_fiu_fuzzer.c.

5.5.2 Macro Definition Documentation

5.5.2.1 SCRATCH_BUFFER_SIZE

```
#define SCRATCH_BUFFER_SIZE 1024
```

Size of the buffer.

5.5.3 Function Documentation

5.5.3.1 fake_open_read()

```
int fake_open_read (
            const char * pathname,
            const pid_t parent_pid )
```

Given a (valid) file path to be open, open instead a PID-dependent file so that concurrent fuzzer executions do not clash.

Parameters

pathname	path of the file to be used
parent_pid	PID of the parent process

Returns

the fid of the multiprocess-safe file opened in this call

5.5.3.2 handle_command_file()

```
void handle_command_file (
            FILE *const file,
            int tool,
            char *const scratch_buffer,
            int(*)(int(*command)(int argc, char *argv[]), int argc, char *argv[], char const
*setstdin, bool closestdout, bool closestderr) call_wrapper )
```

Execute and verify a binary/tool on a file.

Parameters

file	file onto which the command is to be tested
tool	command to be evaluated
scratch_buffer	buffer to be used to parse the call
call_wrapper	function that wraps the actual call to the tool

5.6 command_line_fuzzer_include_helper.h File Reference

```
#include "../bin/v2f_verify_codec.c"
#include <TOOL_FILE>
```

Macros

- #define CONCATENATE2(X, Y) X##Y
- #define CONCATENATE(X, Y) CONCATENATE2(X,Y)
- #define show_usage_string CONCATENATE(show_usage_string_, TOOL_NAME)
- #define main CONCATENATE(main_, TOOL_NAME)

5.6.1 Detailed Description

This file enables #include within #define for command_line_fuzzer.c.

5.6.2 Macro Definition Documentation

5.6.2.1 CONCATENATE

```
#define CONCATENATE(
              X,
              Y ) CONCATENATE2(X,Y)
```

A helper macro for token concatenation.

Parameters

X	first token
Y	second token

5.6.2.2 CONCATENATE2

```
#define CONCATENATE2(
              X,
              Y ) X##Y
```

A helper macro for token concatenation.

Parameters

X	first token
Y	second token

5.6.2.3 main

```
int main CONCATENATE(main_, TOOL_NAME)
```

Macro to get the correct main function for the given tool.

5.6.2.4 show_usage_string

```
#define show_usage_string CONCATENATE(show_usage_string_, TOOL_NAME)
```

Macro to get the correct show_usage_string for the given tool.

5.7 common.c File Reference

```
#include "common.h"
#include <assert.h>
#include "v2f.h"
#include <stdio.h>
```

Functions

- uint32_t v2f_get_bit (uint8_t const ∗const buffer, const uint32_t index)
- void v2f_set_bit (uint8_t ∗const buffer, const uint32_t index, const uint32_t value)
- bool v2f_is_all_zero (uint8_t const ∗const vector, const uint32_t length_bits)
- void debug_show_vector_contents (char ∗name, uint8_t ∗vector, uint32_t vector_length_bits)

5.7.1 Detailed Description

This file implements functions common to the encoder and the decoder.

See also

v2f_common.h for additional details.

5.7.2 Function Documentation

5.7.2.1 debug_show_vector_contents()

```
void debug_show_vector_contents (
            char * name,
            uint8_t * vector,
            uint32_t vector_length_bits )
```

Show the contents of a vector

Parameters

name	a name identifying the vector contents
vector	vector to be analyzed
vector_length_bits	lenght of the vector in bits

5.7.2.2 v2f_get_bit()

```
uint32_t v2f_get_bit (
            uint8_t const *const buffer,
            const uint32_t index )
```

Read the value of a single bit in *buffer*, at a given *index* position.

Note that *index* 0 indicates the MSB of the first byte.

Parameters

buffer	buffer to be read.
index	position within the buffer.

Returns

0 or 1, depending on the value of the selected bit.

5.7.2.3 v2f_is_all_zero()

```
bool v2f_is_all_zero (
            uint8_t const *const vector,
            const uint32_t length_bits )
```

Check whether the first *length_bits* bits of *vector* are all zero.

Parameters

vector	vector to be checked
length_bits	number of bits to be checked from vector. Must be at least 1.

Returns

true if and only if the vector is identically zero

5.7.2.4 v2f_set_bit()

```
void v2f_set_bit (
            uint8_t *const buffer,
            const uint32_t index,
            const uint32_t value )
```

Set a single bit value in the selected position of buffer.

Note that indexing semantics are identical to those of v2f_get_bit, i.e., *index* 0 indicates the MSB of the first byte.

Parameters

buffer	buffer where the value is to be set.
index	position within buffer where the value is to be set.
value	value to be assigned to the selected position (must be 0 or 1).

5.8 common.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
```

Macros

- #define V2F_SILENCE_UNUSED(x) ((void) (x))
- #define V2F_SILENCE_ONLY_USED_BY_ASSERT(x)

Functions

- uint32_t v2f_get_bit (uint8_t const *const buffer, const uint32_t index)
- void v2f_set_bit (uint8_t *const buffer, const uint32_t index, const uint32_t value)
- bool v2f_is_all_zero (uint8_t const *const vector, const uint32_t length_bits)
- void debug_show_vector_contents (char *name, uint8_t *vector, uint32_t vector_length_bits)

5.8.1 Detailed Description

This file exports a few miscellaneous functions and macros.

Among the exported functions there are redefinitions of the abs function in stdlib.h to make sure it has the proper data width, or definitions of minimum and maximum functions as, even if they are usually provided, they are not in C99.

5.8.2 Macro Definition Documentation

5.8.2.1 V2F_SILENCE_ONLY_USED_BY_ASSERT

```
#define V2F_SILENCE_ONLY_USED_BY_ASSERT(
              x )
```

Silences an unused variable warning when variable is only used by an assert. This macro shall be employed when code is complied with NDEBUG defined to disable warnings resulting from variables that are only used an assertion. As the assertion is disabled, an unused variable appears.

This macro does nothing when NDEBUG is defined.

Example:

```
void function(int min, int max) {
    int difference = max - min;
    assert(difference > 0);
    V2F_SILENCE_ONLY_USED_BY_ASSERT(difference);
}
```

Parameters

x	variable to silence

5.8.2.2 V2F_SILENCE_UNUSED

```
#define V2F_SILENCE_UNUSED(
              x ) ((void) (x))
```

Silences an unused variable warning. This macro shall be employed when a function for some reason must have an argument that must not use.

Example:
```
void function(int unused) {
    V2F_SILENCE_UNUSED(unused);
}
```

Parameters

x	variable to silence

5.8.3 Function Documentation

5.8.3.1 debug_show_vector_contents()

```
void debug_show_vector_contents (
            char * name,
            uint8_t * vector,
            uint32_t vector_length_bits )
```

Show the contents of a vector

Parameters

name	a name identifying the vector contents
vector	vector to be analyzed
vector_length_bits	lenght of the vector in bits

5.8.3.2 v2f_get_bit()

```
uint32_t v2f_get_bit (
            uint8_t const *const buffer,
            const uint32_t index )
```

Read the value of a single bit in *buffer*, at a given *index* position.

Note that *index* 0 indicates the MSB of the first byte.

Parameters

buffer	buffer to be read.
index	position within the buffer.

Returns

0 or 1, depending on the value of the selected bit.

5.8.3.3 v2f_is_all_zero()

```
bool v2f_is_all_zero (
            uint8_t const *const vector,
            const uint32_t length_bits )
```

Check whether the first *length_bits* bits of *vector* are all zero.

Parameters

vector	vector to be checked
length_bits	number of bits to be checked from vector. Must be at least 1.

Returns

true if and only if the vector is identically zero

5.8.3.4 v2f_set_bit()

```
void v2f_set_bit (
            uint8_t *const buffer,
            const uint32_t index,
            const uint32_t value )
```

Set a single bit value in the selected position of buffer.

Note that indexing semantics are identical to those of v2f_get_bit, i.e., *index* 0 indicates the MSB of the first byte.

Parameters

buffer	buffer where the value is to be set.
index	position within buffer where the value is to be set.
value	value to be assigned to the selected position (must be 0 or 1).

5.9 common_test.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "../src/common.h"
#include "CUExtension.h"
#include "suite_registration.h"
#include "test_common.h"
#include "test_samples.h"
```

5.9.1 Detailed Description

Suite of tests for the common.h module.

5.10 compressor_test.c File Reference

```
#include <stdio.h>
#include <unistd.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/timer.h"
#include "../src/log.h"
#include "../src/v2f_compressor.h"
#include "../src/v2f_decompressor.h"
#include "../src/v2f_entropy_coder.h"
#include "../src/v2f_entropy_decoder.h"
#include "../src/v2f_build.h"
```

Functions

- void test_compressor_create (void)
- void test_compression_decompression_steps (void)
- void test_compression_decompression_minimal_codec (void)
- void register_compressor (void)

5.10.1 Detailed Description

Test suite for the compressor.

5.10.2 Function Documentation

5.10.2.1 register_compressor()

```
void register_compressor (
            void  )
```

Register the compressor suite

5.10.2.2 test_compression_decompression_minimal_codec()

```
void test_compression_decompression_minimal_codec (
            void  )
```

Test the compression/decompression cycle with v2f_compressor_t and v2f_decompressor_t structures.

Requirement V2F-1.1, V2F-1.2, V2F-1.3, V2F-1.4

5.10.2.3 test_compression_decompression_steps()

```
void test_compression_decompression_steps (
            void  )
```

Test the compression/decompression cycle step by step, testing several parameters of the pipeline and a minimal entropy codec.

This exercises all quantization step and decorrelation mode combinations with a minimal V2F entropy coder.

Requirement V2F-1.1, V2F-1.2, V2F-1.3, V2F-1.4, V2F-2.1

5.10.2.4 test_compressor_create()

```
void test_compressor_create (
            void  )
```

Exercise compressor creation

Requirement V2F-1.3

5.11 CUExtension.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fenv.h>
#include <CUnit/CUError.h>
```

Macros

- #define CU_START_REGISTRATION(name)
- #define CU_QADD_TEST(test)
- #define CU_END_REGISTRATION() }
- #define FAIL_IF_FAIL(x)

5.11.1 Detailed Description

Extension to the CUnit library to allow simplified test suite declaration/registration and automatic floating point exception checking in all tests.

Usage:

```
CU_START_REGISTRATION(suite_name)
CU_QADD_TEST(test_function_1)
CU_QADD_TEST(test_function_2)
// ...
CU_QADD_TEST(test_function_N)
CU_END_REGISTRATION()
```

5.11.2 Macro Definition Documentation

5.11.2.1 CU_END_REGISTRATION

```
#define CU_END_REGISTRATION( ) }
```

Finish the registration of the current test suite

5.11.2.2 CU_QADD_TEST

```
#define CU_QADD_TEST(
              test )
```

Value:

```
{ \
CU_pTest pTest = CU_add_test(pSuite, #test, test); \
if (! pTest) { fprintf(stderr, "%s:%d: %s\n", __FILE__, __LINE__, CU_get_error_msg()); exit(EXIT_FAILURE); }
       \
}
```

Add a test to a test suite.

Must appear between CU_START_REGISTRATION and CU_END_REGISTRATION

Parameters

test	test function to add to the current suite

5.11.2.3 CU_START_REGISTRATION

```
#define CU_START_REGISTRATION(
              name )
```

Value:

```
void register_ ## name (void); \
void register_ ## name (void) { \
CU_pSuite pSuite = CU_add_suite(#name, NULL, NULL); \
if (!pSuite) { fprintf(stderr, "%s: %s\n", __FILE__, CU_get_error_msg()); exit(EXIT_FAILURE); }
```

Register a test suite.

Note

The *register_<test_suite_name>* function must have been defined in suite_registration.h

Parameters

name	name of the test suite to register

5.11.2.4 FAIL_IF_FAIL

```
#define FAIL_IF_FAIL(
                x )
```

Value:

```
do { v2f_error_t return_value = (x); \
if (return_value != V2F_E_NONE) { printf("FAIL_IF_FAIL: Error =  %d\n", return_value);} \
CU_ASSERT_EQUAL_FATAL(return_value, V2F_E_NONE) } while(0)
```

Like RETURN_IF_FAIL, but fatally fails a test.

5.12 decorrelator_test.c File Reference

```
#include <stdio.h>
#include <unistd.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/timer.h"
#include "../src/log.h"
#include "../src/v2f_decorrelator.h"
```

Functions

- void test_decorrelator_create (void)
- void test_decorrelator_prediction_mapping (void)
- void register_decorrelator (void)

5.12.1 Detailed Description

Test suite for the decorrelator.

5.12.2 Function Documentation

5.12.2.1 register_decorrelator()

```
void register_decorrelator (
            void  )
```

Register the decorrelation suite

5.12.2.2 test_decorrelator_create()

```
void test_decorrelator_create (
            void  )
```

Exercise decorrelator creation.

Requirement V2F-1.1

5.12.2.3 test_decorrelator_prediction_mapping()

```
void test_decorrelator_prediction_mapping (
            void  )
```

Test the method for mapping positive and negative prediction errors without expanding the dynamic range.

- **Requirement** V2F-1.1.2

5.13 entropy_codec_test.c File Reference

```
#include <stdio.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/v2f_build.h"
#include "../src/timer.h"
```

Functions

- void test_create_destroy (void)
- void test_coder_basic (void)
- void register_entropy_codec (void)

5.13.1 Detailed Description

Unit tests for the coder.

5.13.2 Function Documentation

5.13.2.1 register_entropy_codec()

```
void register_entropy_codec (
            void  )
```

Register the entropy codec (coder and decoder) suite

5.13.2.2 test_coder_basic()

```
void test_coder_basic (
            void  )
```

Test basic coding and decoding with minimal V2F coder/decoder pairs.

5.13.2.3 test_create_destroy()

```
void test_create_destroy (
            void  )
```

Evaluate parameter handling in the coder initialization and destruction.

5.14 errors.c File Reference

```
#include "errors.h"
#include <assert.h>
#include "v2f.h"
```

Functions

- const char * v2f_strerror (v2f_error_t error)

Variables

- static const char *const v2f_error_strings [ ]

5.14.1 Detailed Description

Implementation of the interface for consistent error management.

See also

errors.h for further details and usage information.

5.14.2 Function Documentation

5.14.2.1 v2f_strerror()

```
const char* v2f_strerror (
            v2f_error_t error )
```

Return a string representing the v2f_error_t passed as argument. It must be a valid error value.

Parameters

error	error to be represented

Returns

a string

5.14.3 Variable Documentation

5.14.3.1 v2f_error_strings

```
const char* const v2f_error_strings[]  [static]
```

Initial value:

```
= {
        "V2F_E_NONE",
        "V2F_E_UNEXPECTED_END_OF_FILE",
        "V2F_E_IO",
        "V2F_E_CORRUPTED_DATA",
        "V2F_E_INVALID_PARAMETER",
        "V2F_E_NON_ZERO_RESERVED_OR_PADDING",
        "V2F_E_UNABLE_TO_CREATE_TEMPORARY_FILE",
        "V2F_E_OUT_OF_MEMORY",
        "V2F_E_FEATURE_NOT_IMPLEMENTED",
}
```

Table to convert from an v2f_error_t constant value to its literal representation.

5.15 errors.h File Reference

```
#include <stdint.h>
#include "v2f.h"
```

Macros

- #define RETURN_IF_FAIL(x)

Functions

- const char * v2f_strerror (v2f_error_t error)

5.15.1 Detailed Description

Interface for consistent error management.

Internal-facing functions may assert() or abort() for causes of incorrect function usage. However, functions of the public API (i.e., those declared as *V2F_EXPORTED_SYMBOL* in v2f.h) shall never crash. Instead, they should return an error that can be handled externally.

The protocol used in this implementation to achieve this goal is the definition of v2f_error_t as return type for most internal and external functions, plus the use of the RETURN_IF_FAIL macro to provide an exception-like behavior in case an error is detected. On success, functions are expected to return V2F_E_NONE.

Usage:
```
v2f_error_t example_function(...) {
   // Both function1 and function2 conform to the aforementioned protocol
   // and have v2f_error_t return type
   RETURN_IF_FAIL(function1(...));
   RETURN_IF_FAIL(function2(...));
   return V2F_E_NONE;
}
```

5.15.2 Macro Definition Documentation

5.15.2.1 RETURN_IF_FAIL

```
#define RETURN_IF_FAIL(
               x )
```

Value:
```
    do { const v2f_error_t local_return_value = (x); \
    if(local_return_value != V2F_E_NONE) return local_return_value; } while(0)
```

Return from the current function if a given v2f_error_t value is not V2F_E_NONE.

Given a function v2f_error_t f(...) usage is as follows:
```
RETURN_IF_FAIL(f(...));
```

This is equivalent to writing:
```
{
    v2f_error_t return_value = f(..);
    if (return_value != V2F_E_NONE) {
        return return_value;
    }
}
```

Parameters

x	v2f_error_t value to be evaluated.

5.15.3 Function Documentation

5.15.3.1 v2f_strerror()

```
const char* v2f_strerror (
            v2f_error_t error )
```

Return a string representing the v2f_error_t passed as argument. It must be a valid error value.

Parameters

error	error to be represented

Returns

a string

5.16 file_test.c File Reference

```
#include <stdio.h>
#include <unistd.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/timer.h"
#include "../src/log.h"
#include "../src/v2f_file.h"
#include "../src/v2f_build.h"
#include "test_samples.h"
```

Functions

- void test_sample_io (void)
- void test_minimal_forest_dump (void)
- void test_minimal_codec_dump (void)
- void register_file (void)

5.16.1 Detailed Description

Test suite for the file interface module.

5.16.2 Function Documentation

5.16.2.1 register_file()

```
void register_file (
            void  )
```

Register the file suite

5.16.2.2 test_minimal_codec_dump()

```
void test_minimal_codec_dump (
            void  )
```

Test that compressor/decompressor pairs (codecs) can be properly dumped and loaded

5.16.2.3 test_minimal_forest_dump()

```
void test_minimal_forest_dump (
            void  )
```

Test that entropy coders/decoders can be properly dumped and loaded

5.16.2.4 test_sample_io()

```
void test_sample_io (
            void  )
```

Exercise file I/O.

5.17 fuzzer_compress_decompress.c File Reference

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include "../src/v2f.h"
#include "../src/v2f_file.h"
#include "../src/common.h"
#include "../src/log.h"
#include "fuzzing_common.h"
```

Macros

- #define MIN_HEADER_NAME_SIZE 6

Functions

- void run_one_case (FILE *samples_file, FILE *header_file, FILE *compressed_file, FILE *reconstructed_↩ file)
- static bool is_regular_file (const char *path)
- int main (int argc, char *argv[ ])

5.17.1 Detailed Description

This fuzzer:

1. Reads an input file, codec definition and other function parameters.
2. Attempts compression.
3. Upon successful compression, decompression is attempted.
4. If lossless reconstruction is expected, it is asserted.

The expected input format is:

1. Number of bytes in the sample file: 4 bytes, unsigned big endian
2. Number of chars of the header filename: 2 bytes, unsigned big endian
3. Header name string of exactly that length, without trailing '\0'
4. Samples: with length in bytes as defined in 1

5.17.2 Macro Definition Documentation

5.17.2.1 MIN_HEADER_NAME_SIZE

```
#define MIN_HEADER_NAME_SIZE 6
```

Minimum length allowed in header sample paths.

5.17.3 Function Documentation

5.17.3.1 is_regular_file()

```
static bool is_regular_file (
            const char * path )  [static]
```

Determine whether a given path is a directory. We don't want to open those.

Parameters

path	path to be verified

Returns

true if and only if the path points to a directory

5.17.3.2 main()

```
int main (
            int argc,
            char * argv[] )
```

Entry point for the fuzzer harness.

Parameters

argc	number of command line arguments
argv	command line arguments

Returns

the return code

5.17.3.3 run_one_case()

```
void run_one_case (
            FILE * samples_file,
            FILE * header_file,
            FILE * compressed_file,
            FILE * reconstructed_file )
```

Run a single decoding instance from samples_file with a minimal entropy coder.

Parameters

samples_file	file open for reading with samples to be compressed
header_file	file open for reading with the V2F codec definition to use
compressed_file	file open for reading and writing where compressed data are stored and then read for decompression.
reconstructed_file	file open for reading and writing where the reconstructed data are stored.

5.18 fuzzer_entropy_coder.c File Reference

```
#include <stdio.h>
#include <assert.h>
```

```
#include <string.h>
#include "../src/v2f.h"
#include "../src/v2f_build.h"
#include "../src/v2f_file.h"
#include "../src/common.h"
#include "fuzzing_common.h"
```

Functions

- void run_one_case (FILE *samples_file, uint32_t sample_count, uint8_t bytes_per_sample, v2f_sample_t *const input_samples, uint8_t *const compressed_bytes, v2f_sample_t *const reconstructed_samples)
- int main (int argc, char *argv[])

5.18.1 Detailed Description

Fuzzer harness that exercises the entropy coding stage.

5.18.2 Function Documentation

5.18.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Entry point for the fuzzer harness.

Parameters

argc	number of command line arguments
argv	command line arguments

Returns

the return code

5.18.2.2 run_one_case()

```
void run_one_case (
            FILE * samples_file,
            uint32_t sample_count,
```

```
            uint8_t bytes_per_sample,
            v2f_sample_t *const input_samples,
            uint8_t *const compressed_bytes,
            v2f_sample_t *const reconstructed_samples )
```

Run a single decoding instance from samples_file with a minimal entropy coder.

Parameters

samples_file	file open for reading with the data to be compressed
sample_count	number of samples to be read
bytes_per_sample	number of bytes per samples
input_samples	pre-allocated array of samples_file samples
compressed_bytes	pre-allocated array of compressed bytes
reconstructed_samples	pre-allocated array of reconstructed samples.

5.19 fuzzer_entropy_decoder.c File Reference

```
#include <stdio.h>
#include <assert.h>
#include <unistd.h>
#include <string.h>
#include "../src/v2f.h"
#include "../src/v2f_build.h"
#include "../src/v2f_file.h"
#include "../src/common.h"
#include "fuzzing_common.h"
```

Functions

- void run_one_case (FILE *input, uint32_t sample_count, uint8_t bytes_per_word, v2f_sample_t *const sample_buffer, uint8_t *const compressed_buffer)
- int main (int argc, char *argv[])

5.19.1 Detailed Description

Fuzzer harness that exercises the entropy decoding stage.

5.19.2 Function Documentation

5.19.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Entropy point to the fuzzer's harness.

Parameters

argc	number of command line arguments
argv	command line arguments

Returns

status code

5.19.2.2 run_one_case()

```
void run_one_case (
            FILE * input,
            uint32_t sample_count,
            uint8_t bytes_per_word,
            v2f_sample_t *const sample_buffer,
            uint8_t *const compressed_buffer )
```

Run a single decoding instance from input

Parameters

input	file open for reading with compressed data to be decoded
sample_count	number of samples to be decoded
bytes_per_word	number of bytes per compressed codeword
sample_buffer	pre-allocated buffer to store the decodes samples
compressed_buffer	buffer of compressed codewords to be decoded

5.20 fuzzing_common.c File Reference

```
#include <stdbool.h>
#include <unistd.h>
#include <assert.h>
#include "fuzzing_common.h"
#include "../src/v2f.h"
#include "../src/common.h"
#include "../src/errors.h"
```

Functions

- void fuzzing_reset_file (FILE *const file)
- bool fuzzing_check_files_are_equal (FILE *f1, FILE *f2)
- void copy_file (FILE *input, FILE *output)
- v2f_error_t v2f_fuzzing_assert_temp_file_created (FILE **const temporary_file)
- v2f_error_t fuzzing_get_samples_and_bytes_per_sample (FILE *file, uint32_t *sample_count, uint8_t *bytes_per_sample)

5.20.1 Detailed Description

Implementation of utility functions common to several fuzzers.

I/O errors in this module are tagged with LCOV_EXCL_LINE, since they are not meant to be triggered.

5.20.2 Function Documentation

5.20.2.1 copy_file()

```
void copy_file (
            FILE * input,
            FILE * output )
```

Copy the remaining data of *input* into *output* and move the file pointer of *output* to the first byte of the file.

Parameters

input	file to be copied.
output	file where the copy is to be stored.

5.20.2.2 fuzzing_check_files_are_equal()

```
bool fuzzing_check_files_are_equal (
            FILE * f1,
            FILE * f2 )
```

Verify whether two files open for reading have equal (remaining) size and contents.

Parameters

f1	first file to be compared.
f2	second file to be compared.

Returns

true if and only if file are bitwise identical.

5.20.2.3 fuzzing_get_samples_and_bytes_per_sample()

```
v2f_error_t fuzzing_get_samples_and_bytes_per_sample (
            FILE * file,
```

```
            uint32_t * sample_count,
            uint8_t * bytes_per_sample )
```

Read the sample count and bytes per index from the input file.

- samples: 4 byte unsigned big endian
- bytes per sample/index: 1 byte

Parameters

file	file open for reading
sample_count	pointer to the variable where the number of samples is to be stored
bytes_per_sample	pointer to the variable where the number of bytes per sample is to be stored.

Returns

- V2F_E_NONE : values successfully read
- V2F_E_CORRUPTED_DATA : invalid values read
- V2F_E_IO : input/output error

5.20.2.4 fuzzing_reset_file()

```
void fuzzing_reset_file (
            FILE *const file )
```

Truncate a file to zero bytes and set the file pointer to the first byte.

Parameters

file	file to be truncated.

5.20.2.5 v2f_fuzzing_assert_temp_file_created()

```
v2f_error_t v2f_fuzzing_assert_temp_file_created (
            FILE **const temporary_file )
```

Generate a temporary file asserting that it is properly open.

Parameters

temporary_file	pointer to the FILE* pointer that should point to the newly created file.

Returns

always V2F_E_NONE, otherwise it aborts.

5.21 fuzzing_common.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "../src/errors.h"
```

Macros

- #define __AFL_LOOP(x) !(count++)
- #define ABORT_IF_FAIL(x)

Functions

- void fuzzing_reset_file (FILE *const file)
- bool fuzzing_check_files_are_equal (FILE *f1, FILE *f2)
- void copy_file (FILE *input, FILE *output)
- v2f_error_t v2f_fuzzing_assert_temp_file_created (FILE **const temporary_file)
- v2f_error_t fuzzing_get_samples_and_bytes_per_sample (FILE *file, uint32_t *sample_count, uint8_t *bytes_per_sample)

Variables

- static int count = 0

5.21.1 Detailed Description

Declaration of utility functions common to several fuzzers.

5.21.2 Macro Definition Documentation

5.21.2.1 __AFL_LOOP

```
#define __AFL_LOOP(
              x ) !(count++)
```

Define a loop-once __AFL_LOOP if not using afl-clang-fast

5.21.2.2 ABORT_IF_FAIL

```
#define ABORT_IF_FAIL(
                x )
```

Value:

```
    do { v2f_error_t return_value = (x); if(return_value != V2F_E_NONE) \
    { printf("Error %s at %s:%d\n", v2f_strerror(return_value), __FILE__, (int) __LINE__); abort(); } }
       while(0)
```

The same as RETURN_IF_FAIL, but aborts instead of returning.

5.21.3 Function Documentation

5.21.3.1 copy_file()

```
void copy_file (
              FILE * input,
              FILE * output )
```

Copy the remaining data of *input* into *output* and move the file pointer of *output* to the first byte of the file.

Parameters

input	file to be copied.
output	file where the copy is to be stored.

5.21.3.2 fuzzing_check_files_are_equal()

```
bool fuzzing_check_files_are_equal (
              FILE * f1,
              FILE * f2 )
```

Verify whether two files open for reading have equal (remaining) size and contents.

Parameters

f1	first file to be compared.
f2	second file to be compared.

Returns

true if and only if file are bitwise identical.

5.21.3.3 fuzzing_get_samples_and_bytes_per_sample()

```
v2f_error_t fuzzing_get_samples_and_bytes_per_sample (
            FILE * file,
            uint32_t * sample_count,
            uint8_t * bytes_per_sample )
```

Read the sample count and bytes per index from the input file.

- samples: 4 byte unsigned big endian
- bytes per sample/index: 1 byte

Parameters

file	file open for reading
sample_count	pointer to the variable where the number of samples is to be stored
bytes_per_sample	pointer to the variable where the number of bytes per sample is to be stored.

Returns

- V2F_E_NONE : values successfully read
- V2F_E_CORRUPTED_DATA : invalid values read
- V2F_E_IO : input/output error

5.21.3.4 fuzzing_reset_file()

```
void fuzzing_reset_file (
            FILE *const file )
```

Truncate a file to zero bytes and set the file pointer to the first byte.

Parameters

file	file to be truncated.

5.21.3.5 v2f_fuzzing_assert_temp_file_created()

```
v2f_error_t v2f_fuzzing_assert_temp_file_created (
            FILE **const temporary_file )
```

Generate a temporary file asserting that it is properly open.

Parameters

temporary_file	pointer to the FILE* pointer that should point to the newly created file.

Returns

always V2F_E_NONE, otherwise it aborts.

5.21.4 Variable Documentation

5.21.4.1 count

```
int count = 0  [static]
```

Definition of the AFL loop counter, used by afl-fuzz - warning is OK.

5.22 log.h File Reference

Macros

- #define LOG_ERROR_LEVEL 1
- #define LOG_WARNING_LEVEL 2
- #define LOG_INFO_LEVEL 3
- #define LOG_DEBUG_LEVEL 4
- #define _LOG_DEFAULT_LEVEL LOG_WARNING_LEVEL
- #define _LOG_LEVEL _LOG_DEFAULT_LEVEL
- #define _SHOW_LOG_MESSAGE(level, ...) do {} while (0)
- #define _SHOW_LOG_NO_DECORATION(level, ...) do {} while (0)
- #define log_error(...) _SHOW_LOG_MESSAGE(LOG_ERROR_LEVEL, "Error", __VA_ARGS__)
- #define log_warning(...) _SHOW_LOG_MESSAGE(LOG_WARNING_LEVEL, "Warning", __VA_ARGS__)
- #define log_info(...) _SHOW_LOG_MESSAGE(LOG_INFO_LEVEL, "Info", __VA_ARGS__)
- #define log_debug(...) _SHOW_LOG_MESSAGE(LOG_DEBUG_LEVEL, "Debug", __VA_ARGS__)
- #define log_no_newline(level, ...) _SHOW_LOG_NO_DECORATION(level, __VA_ARGS__)

5.22.1 Detailed Description

Tools to show messages of different priorities.

They can be configured to disable these messages.

5.22.2 Macro Definition Documentation

5.22.2.1 _LOG_DEFAULT_LEVEL

```
#define _LOG_DEFAULT_LEVEL LOG_WARNING_LEVEL
```

Default warning level.

5.22.2.2 _LOG_LEVEL

```
#define _LOG_LEVEL _LOG_DEFAULT_LEVEL
```

Constant controlling the logging behavior.

Sets the maximum debug level to be printed (higher level: lower priority)

5.22.2.3 _SHOW_LOG_MESSAGE

```
#define _SHOW_LOG_MESSAGE(
              level,
              ...  ) do {} while (0)
```

When debug is disabled, log messages can discarded during compilation.

5.22.2.4 _SHOW_LOG_NO_DECORATION

```
#define _SHOW_LOG_NO_DECORATION(
              level,
              ...  ) do {} while (0)
```

When debug is disabled, log messages without decoration can discarded during compilation

5.22.2.5 log_debug

```
#define log_debug(
              ...  ) _SHOW_LOG_MESSAGE(LOG_DEBUG_LEVEL, "Debug", __VA_ARGS__)
```

Log a message with debug priority.

5.22.2.6 LOG_DEBUG_LEVEL

```
#define LOG_DEBUG_LEVEL 4
```

Debug log level.

5.22.2.7 log_error

```
#define log_error(
              ...  ) _SHOW_LOG_MESSAGE(LOG_ERROR_LEVEL, "Error", __VA_ARGS__)
```

Log a message with error priority.

5.22.2.8 LOG_ERROR_LEVEL

```
#define LOG_ERROR_LEVEL 1
```

Error log level.

5.22.2.9 log_info

```
#define log_info(
              ...  ) _SHOW_LOG_MESSAGE(LOG_INFO_LEVEL, "Info", __VA_ARGS__)
```

Log a message with info priority.

5.22.2.10 LOG_INFO_LEVEL

```
#define LOG_INFO_LEVEL 3
```

Info log level.

5.22.2.11 log_no_newline

```
#define log_no_newline(
              level,
              ...  ) _SHOW_LOG_NO_DECORATION(level, __VA_ARGS__)
```

Log a message with arbitrary priority level and no decoration nor newline.

5.22.2.12 log_warning

```
#define log_warning(
              ...  ) _SHOW_LOG_MESSAGE(LOG_WARNING_LEVEL, "Warning", __VA_ARGS__)
```

Log a message with warning priority.

5.22.2.13 LOG_WARNING_LEVEL

```
#define LOG_WARNING_LEVEL 2
```

Warning log level.

5.23 quantizer_test.c File Reference

```
#include <stdio.h>
#include <unistd.h>
#include "CUExtension.h"
#include "test_common.h"
#include "../src/timer.h"
#include "../src/log.h"
#include "../src/v2f_quantizer.h"
```

Functions

- void test_quantizer_create (void)
- void register_quantizer (void)

5.23.1 Detailed Description

Test suite for the file interface module.

5.23.2 Function Documentation

5.23.2.1 register_quantizer()

```
void register_quantizer (
            void  )
```

Register the quantization suite

5.23.2.2 test_quantizer_create()

```
void test_quantizer_create (
            void  )
```

Exercise quantizer creation.

Requirement V2F-2.1

5.24 suite_registration.h File Reference

Functions

- void register_timer (void)
- void register_build (void)
- void register_entropy_codec (void)
- void register_file (void)
- void register_quantizer (void)
- void register_decorrelator (void)
- void register_compressor (void)

5.24.1 Detailed Description

Support file for CUnit where the suite registration functions are declared.

See also

test.c

5.24.2 Function Documentation

5.24.2.1 register_build()

```
void register_build (
            void  )
```

Register the build suite

5.24.2.2 register_compressor()

```
void register_compressor (
            void  )
```

Register the compressor suite

5.24.2.3 register_decorrelator()

```
void register_decorrelator (
            void  )
```

Register the decorrelation suite

5.24.2.4 register_entropy_codec()

```
void register_entropy_codec (
            void  )
```

Register the entropy codec (coder and decoder) suite

5.24.2.5 register_file()

```
void register_file (
            void  )
```

Register the file suite

5.24.2.6 register_quantizer()

```
void register_quantizer (
            void  )
```

Register the quantization suite

5.24.2.7 register_timer()

```
void register_timer (
            void  )
```

Register the timer suite

5.25 test.c File Reference

```
#include <CUnit/Basic.h>
#include "suite_registration.h"
```

Functions

- int main (void)

5.25.1 Detailed Description

Entry point for all unittest suites:

- General functionality tests:

All tests have been run and verified not to fail.

Usage (from the project root):

```
$ make
$ ./build/unittest
```

5.25.2 Function Documentation

5.25.2.1 main()

```
int main (
            void  )
```

Entropy point for all unittests suites. It executes all test suites and reports the test results, stopping on any failure.

Returns

0 is always returned by this function, but test may abort the program if an error is detected.

Requirement V2F-3.1

5.26 test_common.c File Reference

```
#include "../src/common.h"
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#include "test_common.h"
#include <assert.h>
#include <stdio.h>
#include <unistd.h>
#include "CUExtension.h"
```

Functions

- bool test_vectors_are_equal (uint8_t const *const vector1, uint8_t const *const vector2, const uint32_t large_f)
- bool test_assert_files_are_equal (FILE *const file1, FILE *const file2)
- void test_reset_file (FILE *const file)
- off_t get_file_size (FILE *const file)
- void copy_file (FILE *input, FILE *output)

5.26.1 Detailed Description

Interface implementation of functionality shared by several tests suites.

LCOV_EXCL_LINE is used for I/O error points in this module that are not meant to be triggered.

See also

test_common.h for further details.

5.26.2 Function Documentation

5.26.2.1 copy_file()

```
void copy_file (
            FILE * input,
            FILE * output )
```

Copy the remaining data of *input* into *output* and move the file pointer of *output* to the first byte of the file.

Parameters

input	file to be copied.
output	file where the copy is to be stored.

5.26.2.2 get_file_size()

```
off_t get_file_size (
            FILE *const file )
```

Return the size of file in bytes. The file pointer after calling this function is not modified.

Parameters

file	open file to be queried.

Returns

the size of the file in bytes, or -1 if there is an error.

5.26.2.3 test_assert_files_are_equal()

```
bool test_assert_files_are_equal (
            FILE *const file1,
            FILE *const file2 )
```

Given two FILEs open for reading, return true if and only if both have the same (remaining) length_bits and contain the same data.

File pointers are advanced to the end of the file.

Parameters

file1	first file open for reading.
file2	second file open for reading.

Returns

true if the files have the same length in bits and contain the same data, false otherwise.

5.26.2.4 test_reset_file()

```
void test_reset_file (
            FILE *const file )
```

Given an open file for writing, truncate the file to 0 bytes and move the file pointer to the first byte.

Parameters

file	to be truncated.

5.26.2.5 test_vectors_are_equal()

```
bool test_vectors_are_equal (
            uint8_t const *const vector1,
            uint8_t const *const vector2,
            const uint32_t large_f )
```

Compares two vectors. Padding is not compared.

Parameters

vector1	first vector to compare.
vector2	second vector to compare.
large_f	vector size in bits.

Returns

whether the vectors are equal.

5.27 test_common.h File Reference

```
#include <stdint.h>
#include <stdio.h>
```

```
#include <stdbool.h>
#include <limits.h>
#include "../src/errors.h"
#include "../src/v2f.h"
#include "../src/v2f_entropy_coder.h"
#include "../src/v2f_entropy_decoder.h"
```

Functions

- bool test_vectors_are_equal (uint8_t const *const vector1, uint8_t const *const vector2, const uint32_t large_f)
- bool test_assert_files_are_equal (FILE *const file1, FILE *const file2)
- void test_reset_file (FILE *const file)
- off_t get_file_size (FILE *const file)
- void copy_file (FILE *input, FILE *output)

5.27.1 Detailed Description

Interface definition of functionality shared by several tests suites.

5.27.2 Function Documentation

5.27.2.1 copy_file()

```
void copy_file (
            FILE * input,
            FILE * output )
```

Copy the remaining data of *input* into *output* and move the file pointer of *output* to the first byte of the file.

Parameters

input	file to be copied.
output	file where the copy is to be stored.

5.27.2.2 get_file_size()

```
off_t get_file_size (
            FILE *const file )
```

Return the size of file in bytes. The file pointer after calling this function is not modified.

Parameters

file	open file to be queried.

Returns

the size of the file in bytes, or -1 if there is an error.

5.27.2.3 test_assert_files_are_equal()

```
bool test_assert_files_are_equal (
            FILE *const file1,
            FILE *const file2 )
```

Given two FILEs open for reading, return true if and only if both have the same (remaining) length_bits and contain the same data.

File pointers are advanced to the end of the file.

Parameters

file1	first file open for reading.
file2	second file open for reading.

Returns

true if the files have the same length in bits and contain the same data, false otherwise.

5.27.2.4 test_reset_file()

```
void test_reset_file (
            FILE *const file )
```

Given an open file for writing, truncate the file to 0 bytes and move the file pointer to the first byte.

Parameters

file	to be truncated.

5.27.2.5 test_vectors_are_equal()

```
bool test_vectors_are_equal (
```

```
uint8_t const *const vector1,
uint8_t const *const vector2,
const uint32_t large_f )
```

Compares two vectors. Padding is not compared.

Parameters

vector1	first vector to compare.
vector2	second vector to compare.
large_f	vector size in bits.

Returns

whether the vectors are equal.

5.28 test_samples.c File Reference

```
#include "test_samples.h"
```

Variables

- v2f_test_sample_t all_test_samples [V2F_C_TEST_SAMPLE_COUNT]

5.28.1 Detailed Description

Definition of the list of all available test samples.

See also

test_samples.h for further information.

5.28.2 Variable Documentation

5.28.2.1 all_test_samples

```
v2f_test_sample_t all_test_samples[V2F_C_TEST_SAMPLE_COUNT]
```

List of all v2f_test_sample_t instances.

5.29 test_samples.h File Reference

```
#include <stdint.h>
```

Data Structures

- struct v2f_test_sample_t

Enumerations

- enum { V2F_C_TEST_SAMPLE_COUNT = 8 }

Variables

- v2f_test_sample_t all_test_samples [V2F_C_TEST_SAMPLE_COUNT]

5.29.1 Detailed Description

Tools to automate the testing of test samples.

(automatically generated by metasrc/generate_test_samples.py)

5.29.2 Enumeration Type Documentation

5.29.2.1 anonymous enum

```
anonymous enum
```

Test sample constants.

Enumerator

V2F_C_TEST_SAMPLE_COUNT	Number of samples available for testing.

5.29.3 Variable Documentation

5.29.3.1 all_test_samples

```
v2f_test_sample_t all_test_samples[V2F_C_TEST_SAMPLE_COUNT]
```

List of all v2f_test_sample_t instances.

5.30 timer.c File Reference

```
#include "timer.h"
#include <stdio.h>
#include <stdbool.h>
#include <assert.h>
#include <sys/time.h>
```

Functions

- double timer_get_wall_time ()
- void timer_start (char const *const name)
- void timer_stop (char const *const name)
- double timer_get_cpu_s (char const *const name)
- double timer_get_wall_s (char const *const name)
- void timer_report_csv (FILE *const output_file)
- void timer_report_human (FILE *const output_file)
- void timer_reset ()

Variables

- global_timer_t global_timer = {.entry_count = 0}

5.30.1 Detailed Description

Implementation of the timer tools.

5.30.2 Function Documentation

5.30.2.1 timer_get_cpu_s()

```
double timer_get_cpu_s (
            char const *const name )
```

Get the current or total process time of a started or finished named timer.

Parameters

name	name of the timer

Returns

CPU execution time in seconds, or -1 if the name is not found in global_timer.

5.30.2.2 timer_get_wall_s()

```
double timer_get_wall_s (
            char const *const name )
```

Get the current or total wall time of a started or finished named timer.

Parameters

name	name of the timer

Returns

wall execution time, or -1 if the name is not found in global_timer

5.30.2.3 timer_get_wall_time()

```
double timer_get_wall_time (
            void  )
```

Returns

the current wall time

5.30.2.4 timer_report_csv()

```
void timer_report_csv (
            FILE *const output_file )
```

Report the timer state into *output_file* in CSV format.

Parameters

output_file	file where the report is output (e.g., stdout)

5.30.2.5 timer_report_human()

```
void timer_report_human (
            FILE *const output_file )
```

Report the timer state into *output_file* in a human-readable way

Parameters

output_file	file where the report is output (e.g., stdout)

5.30.2.6 timer_reset()

```
void timer_reset (
            void  )
```

Resets the timer erasing any previous information

5.30.2.7 timer_start()

```
void timer_start (
            char const *const name )
```

Start a named timer. The name must be unique.

Parameters

name	\0 ended string, case sensitive, that identifies this timer. Must have length <= 255.

5.30.2.8 timer_stop()

```
void timer_stop (
            char const *const name )
```

Stop a named timer. The name must have been used.

Parameters

name	\0 ended string, case sensitive, that identifies this timer. Must have length <= 255.

5.30.3 Variable Documentation

5.30.3.1 global_timer

```
global_timer_t global_timer = {.entry_count = 0}
```

Global instance to keep track of named timers.

5.31 timer.h File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdint.h>
#include <time.h>
```

Data Structures

- struct timer_entry_t
- struct global_timer_t

Macros

- #define TIMER_TOLERANCE ((double) 1e-2)
- #define MAX_TIMERS 256
- #define NAME_SIZE 256

Functions

- double timer_get_wall_time (void)
- void timer_start (char const *const name)
- void timer_stop (char const *const name)
- double timer_get_cpu_s (char const *const name)
- double timer_get_wall_s (char const *const name)
- void timer_report_csv (FILE *const output_file)
- void timer_report_human (FILE *const output_file)
- void timer_reset (void)

Variables

- global_timer_t global_timer

5.31.1 Detailed Description

Tools to measure execution time.

5.31.2 Macro Definition Documentation

5.31.2.1 MAX_TIMERS

```
#define MAX_TIMERS 256
```

Maximum number of concurrent timers.

5.31.2.2 NAME_SIZE

```
#define NAME_SIZE 256
```

Maximum name size for each timer.

5.31.2.3 TIMER_TOLERANCE

```
#define TIMER_TOLERANCE ((double) 1e-2)
```

Maximum tolerance stored in the timer, in seconds.

5.31.3 Function Documentation

5.31.3.1 timer_get_cpu_s()

```
double timer_get_cpu_s (
            char const *const name )
```

Get the current or total process time of a started or finished named timer.

Parameters

name	name of the timer

Returns

CPU execution time in seconds, or -1 if the name is not found in global_timer.

5.31.3.2 timer_get_wall_s()

```
double timer_get_wall_s (
            char const *const name )
```

Get the current or total wall time of a started or finished named timer.

Parameters

name	name of the timer

Returns

wall execution time, or -1 if the name is not found in global_timer

5.31.3.3 timer_get_wall_time()

```
double timer_get_wall_time (
            void  )
```

Returns

the current wall time

5.31.3.4 timer_report_csv()

```
void timer_report_csv (
            FILE *const output_file )
```

Report the timer state into *output_file* in CSV format.

Parameters

output_file	file where the report is output (e.g., stdout)

5.31.3.5 timer_report_human()

```
void timer_report_human (
            FILE *const output_file )
```

Report the timer state into *output_file* in a human-readable way

Parameters

output_file	file where the report is output (e.g., stdout)

5.31.3.6 timer_reset()

```
void timer_reset (
            void  )
```

Resets the timer erasing any previous information

5.31.3.7 timer_start()

```
void timer_start (
            char const *const name )
```

Start a named timer. The name must be unique.

Parameters

name	\0 ended string, case sensitive, that identifies this timer. Must have length <= 255.

5.31.3.8 timer_stop()

```
void timer_stop (
            char const *const name )
```

Stop a named timer. The name must have been used.

Parameters

name	\0 ended string, case sensitive, that identifies this timer. Must have length <= 255.

5.31.4 Variable Documentation

5.31.4.1 global_timer

```
global_timer_t global_timer
```

Global instance to keep track of named timers.

5.32 timer_test.c File Reference

```
#include <stdio.h>
#include "CUExtension.h"
#include "test_common.h"
#include "math.h"
#include "../src/timer.h"
```

Functions

- void test_basic_usage (void)
- void test_multiple_count (void)
- void register_timer (void)

5.32.1 Detailed Description

Miscellaneous unit tests.

5.32.2 Function Documentation

5.32.2.1 register_timer()

```
void register_timer (
            void  )
```

Register the timer suite

5.32.2.2 test_basic_usage()

```
void test_basic_usage (
            void  )
```

Test the timer with at most one repetition per label.

5.32.2.3 test_multiple_count()

```
void test_multiple_count (
            void  )
```

Test the timer with multiple repetitions of the same label.

5.33 v2f.h File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include "errors.h"
```

Data Structures

- struct v2f_quantizer_t
- struct v2f_decorrelator_t
- struct v2f_entropy_coder_entry_t
- struct v2f_entropy_coder_t
- struct v2f_entropy_decoder_entry_t
- struct v2f_entropy_decoder_root_t
- struct v2f_entropy_decoder_t
- struct v2f_compressor_t
- struct v2f_decompressor_t

Macros

- #define PROJECT_VERSION "20210801"

Enumerations

Quantizer-related definitions

- enum v2f_quantizer_mode_t { V2F_C_QUANTIZER_MODE_NONE = 0, V2F_C_QUANTIZER_MODE_UNIFORM = 1, V2F_C_QUANTIZER_MODE_COUNT = 2 }
- enum v2f_quantizer_constant_t { V2F_C_QUANTIZER_MODE_MAX_STEP_SIZE = 255 }

Decorrelation-related definitions

- enum v2f_decorrelator_mode_t { V2F_C_DECORRELATOR_MODE_NONE = 0, V2F_C_DECORRELATOR_MODE_LEFT = 1, V2F_C_DECORRELATOR_MODE_2_LEFT = 2, V2F_C_DECORRELATOR_MODE_COUNT = 3 }

File-level operation definitions

- enum v2f_dict_file_constant_t { V2F_C_BYTES_PER_INDEX = 4 }

Global definitions

- #define V2F_SAMPLE_T_MAX UINT32_MAX
- #define V2F_C_MAX_ENTRY_COUNT (UINT32_MAX - 1)
- enum v2f_error_t {
  V2F_E_NONE = 0, V2F_E_UNEXPECTED_END_OF_FILE = 1, V2F_E_IO = 2, V2F_E_CORRUPTED_DATA = 3,
  V2F_E_INVALID_PARAMETER = 4, V2F_E_NON_ZERO_RESERVED_OR_PADDING = 5, V2F_E_UNABLE_TO_CREATE_T = 6, V2F_E_OUT_OF_MEMORY = 7,
  V2F_E_FEATURE_NOT_IMPLEMENTED = 8 }
- enum v2f_entropy_constants_t {
  V2F_C_MIN_BYTES_PER_SAMPLE = 1, V2F_C_MAX_BYTES_PER_SAMPLE = 2, V2F_C_MAX_SAMPLE_VALUE,
  V2F_C_MIN_SIGNED_VALUE = INT32_MIN + 1,
  V2F_C_MAX_SIGNED_VALUE = INT32_MAX, **V2F_C_MIN_BYTES_PER_WORD** = 1, V2F_C_MAX_BYTES_PER_WORD = 2, **V2F_C_MIN_SAMPLE_COUNT** = 1,
  V2F_C_MAX_SAMPLE_COUNT = UINT16_MAX, **V2F_C_MIN_ENTRY_COUNT** = 2, V2F_C_MIN_ROOT_COUNT = 1, V2F_C_MAX_ROOT_COUNT = V2F_C_MAX_SAMPLE_VALUE + 1,
  V2F_C_MAX_CHILD_COUNT = V2F_C_MAX_SAMPLE_VALUE + 1, **V2F_C_MIN_BLOCK_SIZE** = 1,
  V2F_C_MAX_BLOCK_SIZE = 5120 * 256, V2F_C_MAX_COMPRESSED_BLOCK_SIZE }
- typedef uint32_t v2f_sample_t
- typedef int32_t v2f_signed_sample_t

Public functions exported in the generated libraries.

- #define V2F_EXPORTED_SYMBOL
- int v2f_file_compress_from_path (char const *const raw_file_path, char const *const header_file_path, char const *const output_file_path, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_compress_from_file (FILE *raw_file, FILE *header_file, FILE *output_file, bool overwrite_↩ quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_decompress_from_path (char const *const compressed_file_path, char const *const header_↩ file_path, char const *const reconstructed_file_path, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_decompress_from_file (FILE *const compressed_file, FILE *const header_file, FILE *const reconstructed_file, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite↩ _qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator↩ _mode)

5.33.1 Detailed Description

Public interface of the V2F compression library.

This file provides the external interface from the core library in src/ to the external world. These functions are expected to operate without crashing under *any* user provided data. All inputs are validated and proper failure codes are returned in case of error, with the exception of an invalid memory pointers, for which it is not possible to tell whether they point to allocated memory or not.

No memory allocation nor file operation is performed in the functions declared in this file. See *v2f_file.h* for functions operating directly on files.

5.33.2 Macro Definition Documentation

5.33.2.1 PROJECT_VERSION

```
#define PROJECT_VERSION "20210801"
```

Software version number.

5.33.2.2 V2F_C_MAX_ENTRY_COUNT

```
#define V2F_C_MAX_ENTRY_COUNT (UINT32_MAX - 1)
```

Maximum number of entries in a V2F tree or forest.

5.33.2.3 V2F_EXPORTED_SYMBOL

```
#define V2F_EXPORTED_SYMBOL
```

The V2F_EXPORTED_SYMBOL macro limits to symbol visibility so that only relevant functions are exported.

5.33.2.4 V2F_SAMPLE_T_MAX

```
#define V2F_SAMPLE_T_MAX UINT32_MAX
```

Maximum value that can be stored in this type.

5.33.3 Typedef Documentation

5.33.3.1 v2f_sample_t

```
typedef uint32_t v2f_sample_t
```

Unsigned sampled value. Entropy coders/decoders use these to represent data.

5.33.3.2 v2f_signed_sample_t

```
typedef int32_t v2f_signed_sample_t
```

Signed sample value. Decorrelation may produce these as intermediate values before sign coding back to v2f_↩ sample_t. Note that in the general pipeline outside decorrelation, only v2f_sample_t sample values are and should be used.

5.33.4 Enumeration Type Documentation

5.33.4.1 v2f_decorrelator_mode_t

```
enum v2f_decorrelator_mode_t
```

List of defined decorrelation modes.

Enumerator

V2F_C_DECORRELATOR_MODE_NONE	Identity decorrelator, that does not modify the input samples.
V2F_C_DECORRELATOR_MODE_LEFT	DPCM decorrelator of order one, using the sample to the left.
V2F_C_DECORRELATOR_MODE_2_LEFT	DPCM decorrelator of order two, using the average of the two samples to the left
V2F_C_DECORRELATOR_MODE_COUNT	Number of available decorrelation modes.

5.33.4.2 v2f_dict_file_constant_t

`enum v2f_dict_file_constant_t`

Constants related to files representing V2F forests

Enumerator

V2F_C_BYTES_PER_INDEX	Number of bytes per index. Fixed size in the header file for simplicity.

5.33.4.3 v2f_entropy_constants_t

`enum v2f_entropy_constants_t`

Constants related to the entropy coders/decoders, common to the whole application.

Enumerator

V2F_C_MAX_BYTES_PER_SAMPLE	Maximum number of bytes allowed to represent each original sample.
V2F_C_MAX_SAMPLE_VALUE	Maximum supported sample value.
V2F_C_MIN_SIGNED_VALUE	Minimum v2f_signed_sample_t value that can be sign coded and still fit in a v2f_sample_t.
V2F_C_MAX_SIGNED_VALUE	Maximum v2f_signed_sample_t that can be sign coded and still fit in a v2f_sample_t.
V2F_C_MAX_BYTES_PER_WORD	Maximum number of bytes used to represent an output codeword.
V2F_C_MIN_ROOT_COUNT	Minimum number of root entries in a V2F codec.
V2F_C_MAX_ROOT_COUNT	Maximum number of root entries in a V2F codec.
V2F_C_MAX_BLOCK_SIZE	Maximum number of samples allowed in a block.
V2F_C_MAX_COMPRESSED_BLOCK_SIZE	Maximum number of bytes in a compressed block, i.e., one word per sample.

5.33.4.4 v2f_error_t

enum v2f_error_t

Functions in this software may return one of these error codes.

This struct must be kept in sync with the v2f_error_strings variable in errors.c

Enumerator

V2F_E_NONE	Function returned no error.
V2F_E_UNEXPECTED_END_OF_FILE	End of file found unexpectedly.
V2F_E_IO	An error occurred while performing an I/O operation.
V2F_E_CORRUPTED_DATA	File is corrupted or syntactically invalid.
V2F_E_INVALID_PARAMETER	Input parameters are not valid.
V2F_E_NON_ZERO_RESERVED_OR_PADDING	Non-zero value found at reserved/padding positions.
V2F_E_UNABLE_TO_CREATE_TEMPORARY_FILE	An error occurred while creating a temporary file.
V2F_E_OUT_OF_MEMORY	Program ran out of memory
V2F_E_FEATURE_NOT_IMPLEMENTED	An application-specific implementation was requested, but is not implemented.

5.33.4.5 v2f_quantizer_constant_t

enum v2f_quantizer_constant_t

Constants that bound the parameters of the quantizer.

Enumerator

V2F_C_QUANTIZER_MODE_MAX_STEP_SIZE	Maximum quantization step size allowed by this module.

5.33.4.6 v2f_quantizer_mode_t

enum v2f_quantizer_mode_t

Types of quantization defined.

Enumerator

V2F_C_QUANTIZER_MODE_NONE	Null quantizer, that does not modify the data.
V2F_C_QUANTIZER_MODE_UNIFORM	Uniform scalar quantizer.
V2F_C_QUANTIZER_MODE_COUNT	Number of quantizer modes available.

5.33.5 Function Documentation

5.33.5.1 v2f_file_compress_from_file()

```
int v2f_file_compress_from_file (
            FILE * raw_file,
            FILE * header_file,
            FILE * output_file,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Compresses an open file into another, using an open header file. It is otherwise identical in behavior to v2f_file_compress_from_path.

Please refer to the user manual for additional information on file formats and other details.

Parameters

raw_file	file open for reading with the data to be read. All remaining data are consumed.
header_file	file open for reading with the V2F codec defintion
output_file	file open for writing where the compressed data is to be stored.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for compression. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for quantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during compression. Otherwise, it is ignored.

Returns

0 if and only if compression was successful

5.33.5.2 v2f_file_compress_from_path()

```
int v2f_file_compress_from_path (
            char const *const raw_file_path,
            char const *const header_file_path,
            char const *const output_file_path,
```

```
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Compress *raw_file_path* into *utput_file_path* using the V2F codec configuration defined in *output_file_path*. Part of this configuration can be overwriten with the provided parameters.

The contents of *header_file_path* are not included in the compressed file. To decompress it, a copy of *header_file_path* must be available at the decoder side, along with the knowledge of any overwriten parameters.

Please refer to the user manual for additional information on file formats and other details.

Parameters

raw_file_path	path to the file with the raw data to compress.
header_file_path	path to the (typically .v2fc) file with the codec definition.
output_file_path	path where the compressed data are to be stored.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for compression. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for quantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during compression. Otherwise, it is ignored.

Returns

0 if and only if compression was successful.

5.33.5.3 v2f_file_decompress_from_file()

```
int v2f_file_decompress_from_file (
            FILE *const compressed_file,
            FILE *const header_file,
            FILE *const reconstructed_file,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Decompresses *compressed_file* into *reconstructed_file* using the V2F codec defined in *header_file*.

Please refer to the user manual for additional information on file formats and other details.

Parameters

compressed_file	file open for reading with the compressed data.
header_file	file open for reading with the header data.
reconstructed_file	file open for writing where the reconstructed data is to be written.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for dequantization. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for dequantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during decompression. Otherwise, it is ignored.

Returns

0 if and only if decompression was successful.

5.33.5.4 v2f_file_decompress_from_path()

```
int v2f_file_decompress_from_path (
            char const *const compressed_file_path,
            char const *const header_file_path,
            char const *const reconstructed_file_path,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Decompress a file *compressed_file_path* produced by v2f_file_compress_from_path, using the compressor configuration given in *header_file_path*. The reconstructed data are stored into *reconstructed_file_path*.

The quantization and decorrelation parameters defined in *header_file_path* can be overwritten using the parameters to this function.

Please refer to the user manual for additional information on file formats and other details.

Parameters

compressed_file_path	path to the compressed bitstream to decompress.
header_file_path	path to a copy of the header file used for compression.
reconstructed_file_path	
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for dequantization. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.

Parameters

step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for dequantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during decompression. Otherwise, it is ignored.

Returns

0 if and only if decompression was successful.

5.34 v2f_build.c File Reference

```
#include "v2f_build.h"
#include <stdlib.h>
#include <assert.h>
#include "log.h"
#include "errors.h"
```

Functions

- v2f_error_t v2f_build_minimal_codec (uint8_t bytes_per_word, v2f_compressor_t ∗compressor, v2f_decompressor_t ∗decompressor)
- v2f_error_t v2f_build_destroy_minimal_codec (v2f_compressor_t ∗compressor, v2f_decompressor_t ∗decompressor)
- v2f_error_t v2f_build_minimal_forest (uint8_t bytes_per_word, v2f_entropy_coder_t ∗coder, v2f_entropy_decoder_t ∗decoder)
- v2f_error_t v2f_build_destroy_minimal_forest (v2f_entropy_coder_t ∗coder, v2f_entropy_decoder_t ∗decoder)

5.34.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

17/02/2021

Tools to build coders and decoders.

5.34.2 Function Documentation

5.34.2.1 v2f_build_destroy_minimal_codec()

```
v2f_error_t v2f_build_destroy_minimal_codec (
            v2f_compressor_t * compressor,
            v2f_decompressor_t * decompressor )
```

Free all memory allocated by v2f_build_minimal_codec.

Parameters

compressor	initialized compressor
decompressor	initialized decompressor

Returns

5.34.2.2 v2f_build_destroy_minimal_forest()

```
v2f_error_t v2f_build_destroy_minimal_forest (
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Release all memory allocated by v2f_build_minimal_forest

Parameters

coder	coder to be destroyed
decoder	pointer to the decoder to be destroyed

Returns

- V2F_E_NONE : Successfully created
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.34.2.3 v2f_build_minimal_codec()

```
v2f_error_t v2f_build_minimal_codec (
            uint8_t bytes_per_word,
            v2f_compressor_t * compressor,
            v2f_decompressor_t * decompressor )
```

Build a minimal compressor/decompressor pair. No quantization nor decorrelation is applied by this pair. The V2F forest is generated by v2f_build_minimal_forest.

Parameters

bytes_per_word	bytes per word to support
compressor	pointer compressor to initialize
decompressor	pointer decompressor to initialize

Returns

5.34.2.4 v2f_build_minimal_forest()

```
v2f_error_t v2f_build_minimal_forest (
            uint8_t bytes_per_word,
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Build a minimal V2F entropy coder/decoder pair with 2^(8∗bytes_per_word) entries, designed to cope with samples up to 2^(8∗bytes_per_word) - 1.

The destroy function should be called to release all employed memory.

Parameters

bytes_per_word	
coder	pointer to the coder to be built
decoder	pointer to the decoder to be built

Returns

- V2F_E_NONE : Successfully created
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.35 v2f_build.h File Reference

```
#include <stdint.h>
#include "v2f.h"
#include "v2f_compressor.h"
#include "v2f_decompressor.h"
```

Enumerations

- enum v2f_build_constant_t { V2F_C_MINIMAL_MIN_BYTES_PER_WORD = V2F_C_MIN_BYTES_PER_↩ WORD, V2F_C_MINIMAL_MAX_BYTES_PER_WORD = 2 }

Functions

- v2f_error_t v2f_build_minimal_codec (uint8_t bytes_per_word, v2f_compressor_t ∗compressor, v2f_decompressor_t ∗decompressor)
- v2f_error_t v2f_build_destroy_minimal_codec (v2f_compressor_t ∗compressor, v2f_decompressor_t ∗decompressor)
- v2f_error_t v2f_build_minimal_forest (uint8_t bytes_per_word, v2f_entropy_coder_t ∗coder, v2f_entropy_decoder_t ∗decoder)
- v2f_error_t v2f_build_destroy_minimal_forest (v2f_entropy_coder_t ∗coder, v2f_entropy_decoder_t ∗decoder)

5.35.1 Detailed Description

Tools to generate coders and decoders.

5.35.2 Enumeration Type Documentation

5.35.2.1 v2f_build_constant_t

```
enum v2f_build_constant_t
```

Constants related to the building of default V2F forests.

Enumerator

V2F_C_MINIMAL_MIN_BYTES_PER_WORD	Minimum number of bytes per sample supported by the v2f_build_minimal_forest function.
V2F_C_MINIMAL_MAX_BYTES_PER_WORD	Maximum number of bytes per sample supported by the v2f_build_minimal_forest function.

5.35.3 Function Documentation

5.35.3.1 v2f_build_destroy_minimal_codec()

```
v2f_error_t v2f_build_destroy_minimal_codec (
            v2f_compressor_t * compressor,
            v2f_decompressor_t * decompressor )
```

Free all memory allocated by v2f_build_minimal_codec.

Parameters

compressor	initialized compressor
decompressor	initialized decompressor

Returns

5.35.3.2 v2f_build_destroy_minimal_forest()

```
v2f_error_t v2f_build_destroy_minimal_forest (
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Release all memory allocated by v2f_build_minimal_forest

Parameters

coder	coder to be destroyed
decoder	pointer to the decoder to be destroyed

Returns

- V2F_E_NONE : Successfully created
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.35.3.3 v2f_build_minimal_codec()

```
v2f_error_t v2f_build_minimal_codec (
            uint8_t bytes_per_word,
            v2f_compressor_t * compressor,
            v2f_decompressor_t * decompressor )
```

Build a minimal compressor/decompressor pair. No quantization nor decorrelation is applied by this pair. The V2F forest is generated by v2f_build_minimal_forest.

Parameters

bytes_per_word	bytes per word to support
compressor	pointer compressor to initialize
decompressor	pointer decompressor to initialize

Returns

5.35.3.4 v2f_build_minimal_forest()

```
v2f_error_t v2f_build_minimal_forest (
            uint8_t bytes_per_word,
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Build a minimal V2F entropy coder/decoder pair with 2^(8∗bytes_per_word) entries, designed to cope with samples up to 2^(8∗bytes_per_word) - 1.

The destroy function should be called to release all employed memory.

Parameters

bytes_per_word	
coder	pointer to the coder to be built
decoder	pointer to the decoder to be built

Returns

- V2F_E_NONE : Successfully created
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.36 v2f_compressor.c File Reference

```
#include "v2f_compressor.h"
#include "timer.h"
```

Functions

- v2f_error_t v2f_compressor_create (v2f_compressor_t ∗compressor, v2f_quantizer_t ∗quantizer, v2f_decorrelator_t ∗decorrelator, v2f_entropy_coder_t ∗entropy_coder)
- v2f_error_t v2f_compressor_compress_block (v2f_compressor_t ∗const compressor, v2f_sample_t ∗const input_samples, uint64_t sample_count, uint8_t ∗const output_buffer, uint64_t ∗const written_byte_count)

5.36.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

08/03/2021

5.36.2 Function Documentation

5.36.2.1 v2f_compressor_compress_block()

```
v2f_error_t v2f_compressor_compress_block (
            v2f_compressor_t *const compressor,
            v2f_sample_t *const input_samples,
            uint64_t sample_count,
            uint8_t *const output_buffer,
            uint64_t *const written_byte_count )
```

Compress the samples in input_samples and write the result to output_buffer using the full pipeline of compressor.

Parameters

compressor	intitialized compressor to be used for compression
input_samples	buffer with at least `input_samples` v2f_sample_t values.
sample_count	number of samples to be coded from the buffer. Must be $<$ UINT64_MAX.
output_buffer	buffer where the output is produced. It must be large enough to accommodate the worst case scenario, i.e., one index is emitted per input symbol (input_samples$*$coder-$>$bytes_per_word bytes).
written_byte_count	pointer to a variable where the number of bytes written to output_buffer is stored. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully compressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.36.2.2 v2f_compressor_create()

```
v2f_error_t v2f_compressor_create (
             v2f_compressor_t * compressor,
             v2f_quantizer_t * quantizer,
             v2f_decorrelator_t * decorrelator,
             v2f_entropy_coder_t * entropy_coder )
```

Initialize a compressor.

Parameters

compressor	compressor to be initialized
quantizer	initialized quantizer
decorrelator	initialized decorrelator
entropy_coder	initialized entropy coder

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.37 v2f_compressor.h File Reference

```
#include "v2f.h"
#include "v2f_quantizer.h"
#include "v2f_decorrelator.h"
#include "v2f_entropy_coder.h"
```

Functions

- v2f_error_t v2f_compressor_create (v2f_compressor_t ∗compressor, v2f_quantizer_t ∗quantizer, v2f_decorrelator_t ∗decorrelator, v2f_entropy_coder_t ∗entropy_coder)
- v2f_error_t v2f_compressor_compress_block (v2f_compressor_t ∗const compressor, v2f_sample_t ∗const input_samples, uint64_t sample_count, uint8_t ∗const output_buffer, uint64_t ∗const written_byte_count)

5.37.1 Detailed Description

Module that implements an interface for applying a complete compression pipeline.

5.37.2 Function Documentation

5.37.2.1 v2f_compressor_compress_block()

```
v2f_error_t v2f_compressor_compress_block (
            v2f_compressor_t *const compressor,
            v2f_sample_t *const input_samples,
            uint64_t sample_count,
            uint8_t *const output_buffer,
            uint64_t *const written_byte_count )
```

Compress the samples in `input_samples` and write the result to output_buffer using the full pipeline of `compressor`.

Parameters

compressor	intitialized compressor to be used for compression
input_samples	buffer with at least `input_samples` v2f_sample_t values.
sample_count	number of samples to be coded from the buffer. Must be < UINT64_MAX.
output_buffer	buffer where the output is produced. It must be large enough to accommodate the worst case scenario, i.e., one index is emitted per input symbol (input_samples∗coder->bytes_per_word bytes).
written_byte_count	pointer to a variable where the number of bytes written to output_buffer is stored. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully compressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.37.2.2 v2f_compressor_create()

```
v2f_error_t v2f_compressor_create (
            v2f_compressor_t * compressor,
```

```
v2f_quantizer_t * quantizer,
v2f_decorrelator_t * decorrelator,
v2f_entropy_coder_t * entropy_coder )
```

Initialize a compressor.

Parameters

compressor	compressor to be initialized
quantizer	initialized quantizer
decorrelator	initialized decorrelator
entropy_coder	initialized entropy coder

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.38 v2f_decompressor.c File Reference

```
#include "v2f_decompressor.h"
#include "timer.h"
#include "log.h"
```

Functions

- v2f_error_t v2f_decompressor_create (v2f_decompressor_t *decompressor, v2f_quantizer_t *quantizer, v2f_decorrelator_t *decorrelator, v2f_entropy_decoder_t *entropy_decoder)
- v2f_error_t v2f_decompressor_decompress_block (v2f_decompressor_t *const decompressor, uint8_t *const compressed_data, uint64_t buffer_size_bytes, uint64_t max_output_sample_count, v2f_sample_t *const reconstructed_samples, uint64_t *const written_sample_count)

5.38.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

08/03/2021

5.38.2 Function Documentation

5.38.2.1 v2f_decompressor_create()

```
v2f_error_t v2f_decompressor_create (
            v2f_decompressor_t * decompressor,
            v2f_quantizer_t * quantizer,
            v2f_decorrelator_t * decorrelator,
            v2f_entropy_decoder_t * entropy_decoder )
```

Initialize a decompressor.

Parameters

decompressor	decompressor to be initialized
quantizer	initialized quantizer
decorrelator	initialized decorrelator
entropy_decoder	initialized entropy decoder

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.38.2.2 v2f_decompressor_decompress_block()

```
v2f_error_t v2f_decompressor_decompress_block (
            v2f_decompressor_t *const decompressor,
            uint8_t *const compressed_data,
            uint64_t buffer_size_bytes,
            uint64_t max_output_sample_count,
            v2f_sample_t *const reconstructed_samples,
            uint64_t *const written_sample_count )
```

Decompress the codewords in `compressed_data` and write the result to `reconstructed_samples` using the full decompression pipeline.

Parameters

decompressor	intitialized decompressor to be used for compression.
compressed_data	buffer with the codewords to be decompressed.
buffer_size_bytes	number of bytes in `compressed_data`.
max_output_sample_count	maximum number of samples that will be written to `reconstructed_samples` (if available).
reconstructed_samples	buffer where the reconstructed samples are to be writen. The caller is responsible to pass a large enough buffer.
written_sample_count	pointer where the number of reconstructed samples is to be written. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : Decompression successfull
- V2F_E_IO : Input/output error
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.39 v2f_decompressor.h File Reference

```
#include "v2f.h"
#include "v2f_quantizer.h"
```

```
#include "v2f_decorrelator.h"
#include "v2f_entropy_decoder.h"
```

Functions

- v2f_error_t v2f_decompressor_create (v2f_decompressor_t *decompressor, v2f_quantizer_t *quantizer, v2f_decorrelator_t *decorrelator, v2f_entropy_decoder_t *entropy_decoder)
- v2f_error_t v2f_decompressor_decompress_block (v2f_decompressor_t *const decompressor, uint8_t *const compressed_data, uint64_t buffer_size_bytes, uint64_t max_output_sample_count, v2f_sample_t *const reconstructed_samples, uint64_t *const written_sample_count)

5.39.1 Detailed Description

Module that implements an interface for applying a complete decompression pipeline.

5.39.2 Function Documentation

5.39.2.1 v2f_decompressor_create()

```
v2f_error_t v2f_decompressor_create (
            v2f_decompressor_t * decompressor,
            v2f_quantizer_t * quantizer,
            v2f_decorrelator_t * decorrelator,
            v2f_entropy_decoder_t * entropy_decoder )
```

Initialize a decompressor.

Parameters

decompressor	decompressor to be initialized
quantizer	initialized quantizer
decorrelator	initialized decorrelator
entropy_decoder	initialized entropy decoder

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.39.2.2 v2f_decompressor_decompress_block()

```
v2f_error_t v2f_decompressor_decompress_block (
            v2f_decompressor_t *const decompressor,
```

```
uint8_t *const compressed_data,
uint64_t buffer_size_bytes,
uint64_t max_output_sample_count,
v2f_sample_t *const reconstructed_samples,
uint64_t *const written_sample_count )
```

Decompress the codewords in `compressed_data` and write the result to `reconstructed_samples` using the full decompression pipeline.

Parameters

decompressor	intitialized decompressor to be used for compression.
compressed_data	buffer with the codewords to be decompressed.
buffer_size_bytes	number of bytes in `compressed_data`.
max_output_sample_count	maximum number of samples that will be written to `reconstructed_samples` (if available).
reconstructed_samples	buffer where the reconstructed samples are to be writen. The caller is responsible to pass a large enough buffer.
written_sample_count	pointer where the number of reconstructed samples is to be written. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : Decompression successfull
- V2F_E_IO : Input/output error
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40 v2f_decorrelator.c File Reference

```
#include "v2f_decorrelator.h"
#include <assert.h>
#include <stdlib.h>
#include "log.h"
```

Functions

- v2f_error_t v2f_decorrelator_create (v2f_decorrelator_t *decorrelator, v2f_decorrelator_mode_t mode, v2f_sample_t max_sample_value)
- v2f_sample_t v2f_decorrelator_map_predicted_sample (v2f_sample_t sample, v2f_sample_t prediction, v2f_sample_t max_sample_value)
- v2f_sample_t v2f_decorrelator_unmap_sample (v2f_sample_t coded_value, v2f_sample_t prediction, v2f_sample_t max_sample_value)
- v2f_error_t v2f_decorrelator_decorrelate_block (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_↩ samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_invert_block (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_apply_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_↩ samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_inverse_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_apply_2_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_inverse_2_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)

5.40.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

08/03/2021

Implementation of the decorrelation interfaces.

5.40.2 Function Documentation

5.40.2.1 v2f_decorrelator_apply_2_left_prediction()

```
v2f_error_t v2f_decorrelator_apply_2_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply DPCM decorrelation using the mean of the two previous samples, with two exceptions:

- The first sample of each row uses prediction 0
- The second sample of each row uses prediction equal to the first sample.

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples to be decorrelated

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.2 v2f_decorrelator_apply_left_prediction()

```
v2f_error_t v2f_decorrelator_apply_left_prediction (
            v2f_decorrelator_t * decorrelator,
```

```
        v2f_sample_t * input_samples,
        uint64_t sample_count )
```

Apply DPCM decorrelation using the immediately previous sample (prediction is 0 for the first sample of the block), and store the prediction errors.

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples to be decorrelated

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.3 v2f_decorrelator_create()

```
v2f_error_t v2f_decorrelator_create (
            v2f_decorrelator_t * decorrelator,
            v2f_decorrelator_mode_t mode,
            v2f_sample_t max_sample_value )
```

Initialize a decorrelator instance.

Parameters

decorrelator	pointer to decorrelator to initialize
mode	mode index that identifies this decorrelator
max_sample_value	max original sample value

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.4 v2f_decorrelator_decorrelate_block()

```
v2f_error_t v2f_decorrelator_decorrelate_block (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply decorrelation to a block of samples.

Parameters

decorrelator	initialized decorrelator
input_samples	buffer of samples to decorrelate
sample_count	number of samples in the buffer

Returns

- V2F_E_NONE : Decorrelation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.5 v2f_decorrelator_inverse_2_left_prediction()

```
v2f_error_t v2f_decorrelator_inverse_2_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to DPCM using the average of the two previous samples (prediction is assumed to have been 0 for the first two samples of the block).

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.6 v2f_decorrelator_inverse_left_prediction()

```
v2f_error_t v2f_decorrelator_inverse_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to DPCM using the immediately previous sample (prediction is assumed to have been 0 for the first sample of the block).

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples

Returns

- V2F_E_NONE : Creation successfull

- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.7 v2f_decorrelator_invert_block()

```
v2f_error_t v2f_decorrelator_invert_block (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to a block of samples.

Parameters

decorrelator	initialized decorrelator
input_samples	buffer of samples to decorrelate
sample_count	number of samples in the buffer

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.40.2.8 v2f_decorrelator_map_predicted_sample()

```
v2f_sample_t v2f_decorrelator_map_predicted_sample (
             v2f_sample_t sample,
             v2f_sample_t prediction,
             v2f_sample_t max_sample_value )
```

Code the prediction error of sample_value given its prediction and maximum possible sample_value. A similar coding mechanism is used as in CCSDS 123.0-B-2 (see http://dx.doi.org/10.1109/MGRS.2020.↩3048443 for more information).

Parameters

sample	sample to be coded
prediction	predicted value for this sample
max_sample_value	maximum possible sample value

Returns

the v2f_sample_t that represents the prediction error.

5.40.2.9 v2f_decorrelator_unmap_sample()

```
v2f_sample_t v2f_decorrelator_unmap_sample (
            v2f_sample_t coded_value,
            v2f_sample_t prediction,
            v2f_sample_t max_sample_value )
```

Decode the mapping applied by v2f_decorrelator_map_predicted_sample given a coded value and the same prediction used by that function.

Parameters

coded_value	value coded by v2f_decorrelator_map_predicted_sample.
prediction	the predicted value for this sample.
max_sample_value	maximum sample value used when coding the value.

Returns

the sample_value passed to v2f_decorrelator_map_predicted_sample.

5.41 v2f_decorrelator.h File Reference

```
#include "v2f.h"
```

Functions

- v2f_error_t v2f_decorrelator_create (v2f_decorrelator_t *decorrelator, v2f_decorrelator_mode_t mode, v2f_sample_t max_sample_value)
- v2f_sample_t v2f_decorrelator_map_predicted_sample (v2f_sample_t sample, v2f_sample_t prediction, v2f_sample_t max_sample_value)
- v2f_sample_t v2f_decorrelator_unmap_sample (v2f_sample_t coded_value, v2f_sample_t prediction, v2f_sample_t max_sample_value)
- v2f_error_t v2f_decorrelator_decorrelate_block (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_↩ samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_invert_block (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_apply_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_↩ samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_inverse_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_apply_2_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)
- v2f_error_t v2f_decorrelator_inverse_2_left_prediction (v2f_decorrelator_t *decorrelator, v2f_sample_t *input_samples, uint64_t sample_count)

5.41.1 Detailed Description

Tools to apply decorrelation to some input data, e.g., prediction.

These methods transform unsigned samples that may represent unsigned values.

5.41.2 Function Documentation

5.41.2.1 v2f_decorrelator_apply_2_left_prediction()

```
v2f_error_t v2f_decorrelator_apply_2_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply DPCM decorrelation using the mean of the two previous samples, with two exceptions:

- The first sample of each row uses prediction 0
- The second sample of each row uses prediction equal to the first sample.

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples to be decorrelated

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.2 v2f_decorrelator_apply_left_prediction()

```
v2f_error_t v2f_decorrelator_apply_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply DPCM decorrelation using the immediately previous sample (prediction is 0 for the first sample of the block), and store the prediction errors.

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples to be decorrelated

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.3 v2f_decorrelator_create()

```
v2f_error_t v2f_decorrelator_create (
            v2f_decorrelator_t * decorrelator,
            v2f_decorrelator_mode_t mode,
            v2f_sample_t max_sample_value )
```

Initialize a decorrelator instance.

Parameters

decorrelator	pointer to decorrelator to initialize
mode	mode index that identifies this decorrelator
max_sample_value	max original sample value

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.4 v2f_decorrelator_decorrelate_block()

```
v2f_error_t v2f_decorrelator_decorrelate_block (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply decorrelation to a block of samples.

Parameters

decorrelator	initialized decorrelator
input_samples	buffer of samples to decorrelate
sample_count	number of samples in the buffer

Returns

- V2F_E_NONE : Decorrelation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.5 v2f_decorrelator_inverse_2_left_prediction()

```
v2f_error_t v2f_decorrelator_inverse_2_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to DPCM using the average of the two previous samples (prediction is assumed to have been 0 for the first two samples of the block).

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.6 v2f_decorrelator_inverse_left_prediction()

```
v2f_error_t v2f_decorrelator_inverse_left_prediction (
            v2f_decorrelator_t * decorrelator,
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to DPCM using the immediately previous sample (prediction is assumed to have been 0 for the first sample of the block).

Parameters

decorrelator	initialized decorrelator
input_samples	data to be decorrelated
sample_count	number of samples

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.7 v2f_decorrelator_invert_block()

```
v2f_error_t v2f_decorrelator_invert_block (
            v2f_decorrelator_t * decorrelator,
```

```
            v2f_sample_t * input_samples,
            uint64_t sample_count )
```

Apply inverse decorrelation to a block of samples.

Parameters

decorrelator	initialized decorrelator
input_samples	buffer of samples to decorrelate
sample_count	number of samples in the buffer

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.41.2.8 v2f_decorrelator_map_predicted_sample()

```
v2f_sample_t v2f_decorrelator_map_predicted_sample (
            v2f_sample_t sample,
            v2f_sample_t prediction,
            v2f_sample_t max_sample_value )
```

Code the prediction error of sample_value given its prediction and maximum possible sample_value. A similar coding mechanism is used as in CCSDS 123.0-B-2 (see http://dx.doi.org/10.1109/MGRS.2020.↩3048443 for more information).

Parameters

sample	sample to be coded
prediction	predicted value for this sample
max_sample_value	maximum possible sample value

Returns

the v2f_sample_t that represents the prediction error.

5.41.2.9 v2f_decorrelator_unmap_sample()

```
v2f_sample_t v2f_decorrelator_unmap_sample (
            v2f_sample_t coded_value,
            v2f_sample_t prediction,
            v2f_sample_t max_sample_value )
```

Decode the mapping applied by v2f_decorrelator_map_predicted_sample given a coded value and the same prediction used by that function.

Parameters

coded_value	value coded by v2f_decorrelator_map_predicted_sample.
prediction	the predicted value for this sample.
max_sample_value	maximum sample value used when coding the value.

Returns

the sample_value passed to v2f_decorrelator_map_predicted_sample.

5.42 v2f_entropy_coder.c File Reference

```
#include "v2f_entropy_coder.h"
#include <assert.h>
#include <stddef.h>
#include <string.h>
#include "v2f.h"
#include "log.h"
```

Functions

- v2f_error_t v2f_entropy_coder_create (v2f_entropy_coder_t *const coder, v2f_sample_t max_expected_↩ value, uint8_t bytes_per_word, v2f_entropy_coder_entry_t **roots, uint32_t root_count)
- v2f_error_t v2f_entropy_coder_destroy (v2f_entropy_coder_t *const coder)
- v2f_error_t v2f_entropy_coder_compress_block (v2f_entropy_coder_t *const coder, v2f_sample_t const *const input_samples, uint64_t sample_count, uint8_t *const output_buffer, uint64_t *const written_byte_↩ count)
- v2f_error_t v2f_entropy_coder_fill_entry (uint8_t bytes_per_index, uint32_t index, v2f_entropy_coder_entry_t *const entry)
- v2f_sample_t v2f_entropy_coder_buffer_to_sample (uint8_t const *const data_buffer, uint8_t bytes_per_↩ sample)
- void v2f_entropy_coder_sample_to_buffer (const v2f_sample_t sample, uint8_t *const data_buffer, uint8_t bytes_per_sample)

5.42.1 Detailed Description

Implementation of the entropy coding routines.

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

16/02/2021

Version

1.0

5.42.2 Function Documentation

5.42.2.1 v2f_entropy_coder_buffer_to_sample()

```
v2f_sample_t v2f_entropy_coder_buffer_to_sample (
            uint8_t const *const data_buffer,
            uint8_t bytes_per_sample )
```

Read a sample value from a buffer of uint8_t values.

The sample may have more than one byte per sample. If so, the value is treated as big endian.

Parameters

data_buffer	buffer of uint8_t values
bytes_per_sample	number of bytes per sample

Returns

the value of the first sample stored in the buffer, as v2f_sample_t.

5.42.2.2 v2f_entropy_coder_compress_block()

```
v2f_error_t v2f_entropy_coder_compress_block (
            v2f_entropy_coder_t *const coder,
            v2f_sample_t const *const input_samples,
            uint64_t sample_count,
            uint8_t *const output_buffer,
            uint64_t *const written_byte_count )
```

Compress the samples in `input_samples` and write the result to output_buffer using `coder`.

Parameters

coder	intitialized entropy coder to be used for compression
input_samples	buffer with at least `input_samples` v2f_sample_t values.
sample_count	number of samples to be coded from the buffer. Must be $<$ UINT64_MAX.
output_buffer	buffer where the output is produced. It must be large enough to accommodate the worst case scenario, i.e., one index is emitted per input symbol (input_samples$*$coder->bytes_per_word bytes).
written_byte_count	pointer to a variable where the number of bytes written to output_buffer is stored. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully compressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.42.2.3 v2f_entropy_coder_create()

```
v2f_error_t v2f_entropy_coder_create (
            v2f_entropy_coder_t *const coder,
            v2f_sample_t max_expected_value,
            uint8_t bytes_per_word,
            v2f_entropy_coder_entry_t ** roots,
            uint32_t root_count )
```

Initialize an coder.

Parameters

coder	pointer to the coder to be initialized.
max_expected_value	maximum expected value of any input sample
bytes_per_word	number of bytes used to store each codeword
roots	pointers to root entries.
root_count	number of root entries

Returns

- V2F_E_NONE : Initialization was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.42.2.4 v2f_entropy_coder_destroy()

```
v2f_error_t v2f_entropy_coder_destroy (
            v2f_entropy_coder_t *const coder )
```

This function does NOT free any variables passed to the initialization.

Parameters

coder	coder to be destroyed

Returns

- V2F_E_NONE : Destroy was successful
- V2F_E_INVALID_PARAMETER : invalid parameter provided (NULL pointer or coder does not seem initialized)

5.42.2.5 v2f_entropy_coder_fill_entry()

```
v2f_error_t v2f_entropy_coder_fill_entry (
            uint8_t bytes_per_index,
            uint32_t index,
            v2f_entropy_coder_entry_t *const entry )
```

Fill the index bytes of `entry` given its index.

Values are stored in big-endian order.

Parameters

bytes_per_index	coder to which the entry belong
entry	entry to be filled. Its word_bytes member must be a buffer with enough bytes given coder->bytes_per_word.
index	index to be assigned to the entry

Returns

- V2F_E_NONE : The entry was successfully filled
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.42.2.6 v2f_entropy_coder_sample_to_buffer()

```
void v2f_entropy_coder_sample_to_buffer (
            const v2f_sample_t sample,
            uint8_t *const data_buffer,
            uint8_t bytes_per_sample )
```

Write a single sample to a uint8_t buffer. Big endian is used when necessary.

Parameters

sample	sample to be written
data_buffer	buffer with enough space to write `bytes_per_sample` bytes
bytes_per_sample	number of bytes to be used to represent the sample.

5.43 v2f_entropy_coder.h File Reference

```
#include "v2f.h"
```

Functions

- v2f_error_t v2f_entropy_coder_create (v2f_entropy_coder_t ∗const coder, v2f_sample_t max_expected_↩ value, uint8_t bytes_per_word, v2f_entropy_coder_entry_t ∗∗roots, uint32_t root_count)
- v2f_error_t v2f_entropy_coder_destroy (v2f_entropy_coder_t ∗const coder)
- v2f_error_t v2f_entropy_coder_compress_block (v2f_entropy_coder_t ∗const coder, v2f_sample_t const ∗const input_samples, uint64_t sample_count, uint8_t ∗const output_buffer, uint64_t ∗const written_byte_↩ count)
- v2f_error_t v2f_entropy_coder_fill_entry (uint8_t bytes_per_index, uint32_t index, v2f_entropy_coder_entry_t ∗const entry)
- v2f_sample_t v2f_entropy_coder_buffer_to_sample (uint8_t const ∗const data_buffer, uint8_t bytes_per_↩ sample)
- void v2f_entropy_coder_sample_to_buffer (const v2f_sample_t sample, uint8_t ∗const data_buffer, uint8_t bytes_per_sample)

5.43.1 Detailed Description

Definition of the v2f encoder structures and methods.

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

16/02/2021

5.43.2 Function Documentation

5.43.2.1 v2f_entropy_coder_buffer_to_sample()

```
v2f_sample_t v2f_entropy_coder_buffer_to_sample (
            uint8_t const *const data_buffer,
            uint8_t bytes_per_sample )
```

Read a sample value from a buffer of uint8_t values.

The sample may have more than one byte per sample. If so, the value is treated as big endian.

Parameters

data_buffer	buffer of uint8_t values
bytes_per_sample	number of bytes per sample

Returns

the value of the first sample stored in the buffer, as v2f_sample_t.

5.43.2.2 v2f_entropy_coder_compress_block()

```
v2f_error_t v2f_entropy_coder_compress_block (
            v2f_entropy_coder_t *const coder,
            v2f_sample_t const *const input_samples,
            uint64_t sample_count,
            uint8_t *const output_buffer,
            uint64_t *const written_byte_count )
```

Compress the samples in `input_samples` and write the result to output_buffer using `coder`.

Parameters

coder	intitialized entropy coder to be used for compression
input_samples	buffer with at least `input_samples` v2f_sample_t values.
sample_count	number of samples to be coded from the buffer. Must be < UINT64_MAX.
output_buffer	buffer where the output is produced. It must be large enough to accommodate the worst case scenario, i.e., one index is emitted per input symbol (input_samples∗coder->bytes_per_word bytes).
written_byte_count	pointer to a variable where the number of bytes written to output_buffer is stored. If the pointer is NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully compressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.43.2.3 v2f_entropy_coder_create()

```
v2f_error_t v2f_entropy_coder_create (
            v2f_entropy_coder_t *const coder,
            v2f_sample_t max_expected_value,
            uint8_t bytes_per_word,
            v2f_entropy_coder_entry_t ** roots,
            uint32_t root_count )
```

Initialize an coder.

Parameters

coder	pointer to the coder to be initialized.
max_expected_value	maximum expected value of any input sample
bytes_per_word	number of bytes used to store each codeword
roots	pointers to root entries.
root_count	number of root entries

Returns

- V2F_E_NONE : Initialization was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.43.2.4 v2f_entropy_coder_destroy()

```
v2f_error_t v2f_entropy_coder_destroy (
            v2f_entropy_coder_t *const coder )
```

This function does NOT free any variables passed to the initialization.

Parameters

coder	coder to be destroyed

Returns

- V2F_E_NONE : Destroy was successful
- V2F_E_INVALID_PARAMETER : invalid parameter provided (NULL pointer or coder does not seem initialized)

5.43.2.5 v2f_entropy_coder_fill_entry()

```
v2f_error_t v2f_entropy_coder_fill_entry (
            uint8_t bytes_per_index,
            uint32_t index,
            v2f_entropy_coder_entry_t *const entry )
```

Fill the index bytes of `entry` given its index.

Values are stored in big-endian order.

Parameters

bytes_per_index	coder to which the entry belong
entry	entry to be filled. Its word_bytes member must be a buffer with enough bytes given coder->bytes_per_word.
index	index to be assigned to the entry

Returns

- V2F_E_NONE : The entry was successfully filled
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.43.2.6 v2f_entropy_coder_sample_to_buffer()

```
void v2f_entropy_coder_sample_to_buffer (
            const v2f_sample_t sample,
            uint8_t *const data_buffer,
            uint8_t bytes_per_sample )
```

Write a single sample to a uint8_t buffer. Big endian is used when necessary.

Parameters

sample	sample to be written
data_buffer	buffer with enough space to write `bytes_per_sample` bytes
bytes_per_sample	number of bytes to be used to represent the sample.

5.44 v2f_entropy_decoder.c File Reference

```
#include "v2f_entropy_decoder.h"
#include <assert.h>
#include <string.h>
#include "log.h"
#include "errors.h"
```

Functions

- v2f_error_t v2f_entropy_decoder_create (v2f_entropy_decoder_t *const decoder, v2f_entropy_decoder_root_t **roots, uint32_t root_count, uint8_t bytes_per_word, uint8_t bytes_per_sample)
- v2f_error_t v2f_entropy_decoder_destroy (v2f_entropy_decoder_t *const decoder)
- v2f_error_t v2f_entropy_decoder_decompress_block (v2f_entropy_decoder_t *const decoder, uint8_t const *const compressed_block, uint64_t compressed_size, v2f_sample_t *const reconstructed_samples, uint64_t max_output_sample_count, uint64_t *const written_sample_count)
- v2f_error_t v2f_entropy_decoder_decode_next_index (v2f_entropy_decoder_t *const decoder, uint8_t const *const compressed_block, v2f_sample_t *const output_samples, uint32_t *const samples_written)

5.44.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

16/02/2021

Implementation of the V2F decoder.

5.44.2 Function Documentation

5.44.2.1 v2f_entropy_decoder_create()

```
v2f_error_t v2f_entropy_decoder_create (
            v2f_entropy_decoder_t *const decoder,
            v2f_entropy_decoder_root_t ** roots,
            uint32_t root_count,
            uint8_t bytes_per_word,
            uint8_t bytes_per_sample )
```

Initialize a decoder with the given table of decoder entries by index.

Parameters

decoder	decoder to be initialized.
roots	list of root nodes.
root_count	number of roots in `roots`.
bytes_per_word	number of bytes used to store each codeword in the compressed data.
bytes_per_sample	number of bytes used to store each decoded sample.

Returns

- V2F_E_NONE : Creation was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.44.2.2 v2f_entropy_decoder_decode_next_index()

```
v2f_error_t v2f_entropy_decoder_decode_next_index (
            v2f_entropy_decoder_t *const decoder,
            uint8_t const *const compressed_block,
            v2f_sample_t *const output_samples,
            uint32_t *const samples_written )
```

Decode the samples corresponding to the first encoded word in compressed_block.

It is assumed that indices were produced by concatenating the bytes produced by v2f_entropy_coder_fill_entry, i.e., using big-endian ordering when applicable.

Parameters

decoder	initialized entropy decoder to be used for decompression
compressed_block	pointer to the next index position
output_samples	buffer of samples with enough capacity to accommodate the largest sequence of samples encoded by any word of the encoder
samples_written	pointer to a variable where the number of decoded samples represented by the index is to be stored. Ignored if NULL.

Returns

- V2F_E_NONE : The index was successfully decoded
- V2F_E_INVALID_PARAMETER : invalid parameter provided
- V2F_E_CORRUPTED_DATA : compressed data contained an invalid index, i.e., an index >= decoder->total_entry_count.

5.44.2.3 v2f_entropy_decoder_decompress_block()

```
v2f_error_t v2f_entropy_decoder_decompress_block (
            v2f_entropy_decoder_t *const decoder,
            uint8_t const *const compressed_block,
            uint64_t compressed_size,
            v2f_sample_t *const reconstructed_samples,
            uint64_t max_output_sample_count,
            uint64_t *const written_sample_count )
```

Decompress a block compressed with v2f_entropy_coder_compress_block using the encoder corresponding to `decoder`.

Note that the last emitted word might code more samples than there were in the original block before compression. To guarantee a perfectly identical reconstructed block and to avoid buffer overflows, see the `max_output_sample_count` argument.

Parameters

decoder	decoder to be used for decompression
compressed_block	buffer of compressed data
compressed_size	number of bytes to be decompressed from `compressed_block`.
reconstructed_samples	pointer to the output sample buffer. It must be large enough to decode all samples in the first `compressed_size` bytes of `compressed_block`.
max_output_sample_count	maximum number of samples that should be output by this method.
written_sample_count	pointer to a counter where the number of output samples is stored. If NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully decompressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.44.2.4 v2f_entropy_decoder_destroy()

```
v2f_error_t v2f_entropy_decoder_destroy (
            v2f_entropy_decoder_t *const decoder )
```

Destroy a decoder, releasing any resources allocated during initialization.

Parameters

decoder	pointer to the decoder to be destroyed.

Returns

- V2F_E_NONE : Destruction was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.45 v2f_entropy_decoder.h File Reference

```
#include "v2f.h"
#include "v2f_entropy_coder.h"
```

Functions

- v2f_error_t v2f_entropy_decoder_create (v2f_entropy_decoder_t ∗const decoder, v2f_entropy_decoder_root_t ∗∗roots, uint32_t root_count, uint8_t bytes_per_word, uint8_t bytes_per_sample)
- v2f_error_t v2f_entropy_decoder_destroy (v2f_entropy_decoder_t ∗const decoder)
- v2f_error_t v2f_entropy_decoder_decompress_block (v2f_entropy_decoder_t ∗const decoder, uint8_t const ∗const compressed_block, uint64_t compressed_size, v2f_sample_t ∗const reconstructed_samples, uint64_t max_output_sample_count, uint64_t ∗const written_sample_count)
- v2f_error_t v2f_entropy_decoder_decode_next_index (v2f_entropy_decoder_t ∗const decoder, uint8_t const ∗const compressed_block, v2f_sample_t ∗const output_samples, uint32_t ∗const samples_written)

5.45.1 Detailed Description

Implementation of the entropy decoding routines.

5.45.2 Function Documentation

5.45.2.1 v2f_entropy_decoder_create()

```
v2f_error_t v2f_entropy_decoder_create (
            v2f_entropy_decoder_t *const decoder,
            v2f_entropy_decoder_root_t ** roots,
            uint32_t root_count,
            uint8_t bytes_per_word,
            uint8_t bytes_per_sample )
```

Initialize a decoder with the given table of decoder entries by index.

Parameters

decoder	decoder to be initialized.
roots	list of root nodes.
root_count	number of roots in `roots`.
bytes_per_word	number of bytes used to store each codeword in the compressed data.
bytes_per_sample	number of bytes used to store each decoded sample.

Returns

- V2F_E_NONE : Creation was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.45.2.2 v2f_entropy_decoder_decode_next_index()

```
v2f_error_t v2f_entropy_decoder_decode_next_index (
            v2f_entropy_decoder_t *const decoder,
            uint8_t const *const compressed_block,
            v2f_sample_t *const output_samples,
            uint32_t *const samples_written )
```

Decode the samples corresponding to the first encoded word in compressed_block.

It is assumed that indices were produced by concatenating the bytes produced by v2f_entropy_coder_fill_entry, i.e., using big-endian ordering when applicable.

Parameters

decoder	initialized entropy decoder to be used for decompression
compressed_block	pointer to the next index position
output_samples	buffer of samples with enough capacity to accommodate the largest sequence of samples encoded by any word of the encoder
samples_written	pointer to a variable where the number of decoded samples represented by the index is to be stored. Ignored if NULL.

Returns

- V2F_E_NONE : The index was successfully decoded
- V2F_E_INVALID_PARAMETER : invalid parameter provided
- V2F_E_CORRUPTED_DATA : compressed data contained an invalid index, i.e., an index >= decoder->total_entry_count.

5.45.2.3 v2f_entropy_decoder_decompress_block()

```
v2f_error_t v2f_entropy_decoder_decompress_block (
            v2f_entropy_decoder_t *const decoder,
```

```
            uint8_t const *const compressed_block,
            uint64_t compressed_size,
            v2f_sample_t *const reconstructed_samples,
            uint64_t max_output_sample_count,
            uint64_t *const written_sample_count )
```

Decompress a block compressed with v2f_entropy_coder_compress_block using the encoder corresponding to `decoder`.

Note that the last emitted word might code more samples than there were in the original block before compression. To guarantee a perfectly identical reconstructed block and to avoid buffer overflows, see the `max_output_sample_count` argument.

Parameters

decoder	decoder to be used for decompression
compressed_block	buffer of compressed data
compressed_size	number of bytes to be decompressed from `compressed_block`.
reconstructed_samples	pointer to the output sample buffer. It must be large enough to decode all samples in the first `compressed_size` bytes of `compressed_block`.
max_output_sample_count	maximum number of samples that should be output by this method.
written_sample_count	pointer to a counter where the number of output samples is stored. If NULL, it is ignored.

Returns

- V2F_E_NONE : The block was successfully decompressed
- V2F_E_INVALID_PARAMETER : invalid parameter provided

5.45.2.4 v2f_entropy_decoder_destroy()

```
v2f_error_t v2f_entropy_decoder_destroy (
            v2f_entropy_decoder_t *const decoder )
```

Destroy a decoder, releasing any resources allocated during initialization.

Parameters

decoder	pointer to the decoder to be destroyed.

Returns

- V2F_E_NONE : Destruction was successful
- V2F_E_INVALID_PARAMETER : At least one parameter was invalid

5.46 v2f_file.c File Reference

```
#include "v2f_file.h"
#include <string.h>
```

```
#include <stdlib.h>
#include <assert.h>
#include "v2f_entropy_coder.h"
#include "v2f_entropy_decoder.h"
#include "log.h"
```

Functions

- v2f_error_t v2f_file_write_codec (FILE *output_file, v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_read_codec (FILE *input_file, v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_destroy_read_codec (v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_write_forest (FILE *output, v2f_entropy_coder_t const *const coder, v2f_entropy_decoder_t const *const decoder, uint32_t different_roots)
- v2f_error_t v2f_file_read_forest (FILE *input, v2f_entropy_coder_t *const coder, v2f_entropy_decoder_t *const decoder)
- v2f_error_t v2f_file_destroy_read_forest (v2f_entropy_coder_t *coder, v2f_entropy_decoder_t *decoder)
- v2f_error_t v2f_verify_forest (v2f_entropy_coder_t *const coder, v2f_entropy_decoder_t *const decoder)
- v2f_error_t v2f_file_read_big_endian (FILE *input_file, v2f_sample_t *sample_buffer, uint64_t max_↩ sample_count, uint8_t bytes_per_sample, uint64_t *read_sample_count)
- v2f_error_t v2f_file_write_big_endian (FILE *output_file, v2f_sample_t *const sample_buffer, uint64_↩ t sample_count, uint8_t bytes_per_sample)
- int v2f_file_compress_from_path (char const *const raw_file_path, char const *const header_file_path, char const *const output_file_path, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_compress_from_file (FILE *raw_file, FILE *header_file, FILE *output_file, bool overwrite_↩ quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_decompress_from_path (char const *const compressed_file_path, char const *const header_↩ file_path, char const *const reconstructed_file_path, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite_qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator_mode)
- int v2f_file_decompress_from_file (FILE *const compressed_file, FILE *const header_file, FILE *const reconstructed_file, bool overwrite_quantizer_mode, v2f_quantizer_mode_t quantizer_mode, bool overwrite↩ _qstep, v2f_sample_t step_size, bool overwrite_decorrelator_mode, v2f_decorrelator_mode_t decorrelator↩ _mode)

5.46.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

17/02/2021

Implementation of tools to handle data in files.

5.46.2 Function Documentation

5.46.2.1 v2f_file_compress_from_file()

```
int v2f_file_compress_from_file (
            FILE * raw_file,
            FILE * header_file,
            FILE * output_file,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Compresses an open file into another, using an open header file. It is otherwise identical in behavior to v2f_file_compress_from_path.

Please refer to the user manual for additional information on file formats and other details.

Parameters

raw_file	file open for reading with the data to be read. All remaining data are consumed.
header_file	file open for reading with the V2F codec defintion
output_file	file open for writing where the compressed data is to be stored.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for compression. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for quantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during compression. Otherwise, it is ignored.

Returns

0 if and only if compression was successful

5.46.2.2 v2f_file_compress_from_path()

```
int v2f_file_compress_from_path (
            char const *const raw_file_path,
            char const *const header_file_path,
            char const *const output_file_path,
```

```
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Compress *raw_file_path* into *utput_file_path* using the V2F codec configuration defined in *output_file_path*. Part of this configuration can be overwriten with the provided parameters.

The contents of *header_file_path* are not included in the compressed file. To decompress it, a copy of *header_file_path* must be available at the decoder side, along with the knowledge of any overwriten parameters.

Please refer to the user manual for additional information on file formats and other details.

Parameters

raw_file_path	path to the file with the raw data to compress.
header_file_path	path to the (typically .v2fc) file with the codec definition.
output_file_path	path where the compressed data are to be stored.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for compression. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for quantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during compression. Otherwise, it is ignored.

Returns

0 if and only if compression was successful.

5.46.2.3 v2f_file_decompress_from_file()

```
int v2f_file_decompress_from_file (
            FILE *const compressed_file,
            FILE *const header_file,
            FILE *const reconstructed_file,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Decompresses *compressed_file* into *reconstructed_file* using the V2F codec defined in *header_file*.

Please refer to the user manual for additional information on file formats and other details.

Parameters

compressed_file	file open for reading with the compressed data.
header_file	file open for reading with the header data.
reconstructed_file	file open for writing where the reconstructed data is to be written.
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for dequantization. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.
step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for dequantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during decompression. Otherwise, it is ignored.

Returns

0 if and only if decompression was successful.

5.46.2.4 v2f_file_decompress_from_path()

```
int v2f_file_decompress_from_path (
            char const *const compressed_file_path,
            char const *const header_file_path,
            char const *const reconstructed_file_path,
            bool overwrite_quantizer_mode,
            v2f_quantizer_mode_t quantizer_mode,
            bool overwrite_qstep,
            v2f_sample_t step_size,
            bool overwrite_decorrelator_mode,
            v2f_decorrelator_mode_t decorrelator_mode )
```

Decompress a file *compressed_file_path* produced by v2f_file_compress_from_path, using the compressor configuration given in *header_file_path*. The reconstructed data are stored into *reconstructed_file_path*.

The quantization and decorrelation parameters defined in *header_file_path* can be overwritten using the parameters to this function.

Please refer to the user manual for additional information on file formats and other details.

Parameters

compressed_file_path	path to the compressed bitstream to decompress.
header_file_path	path to a copy of the header file used for compression.
reconstructed_file_path	
overwrite_quantizer_mode	if true, the quantizer mode defined in *header_file_path* is overwritten.
quantizer_mode	if *overwrite_quantizer_mode* is true, this mode is employed for dequantization. Otherwise, it is ignored.
overwrite_qstep	if true, the quantizer step size defined in *header_file_path* is overwritten.

Parameters

step_size	if *overwrite_qstep* is true, and if the effective quantization mode is not NULL quantization, this is the step size employed for dequantization. Otherwise, it is ignored.
overwrite_decorrelator_mode	if true, the decorrelator mode defined in *header_file_path* is overwritten.
decorrelator_mode	if *overwrite_decorrelator_mode* is true, this is the decorrelation mode empoyed during decompression. Otherwise, it is ignored.

Returns

0 if and only if decompression was successful.

5.46.2.5 v2f_file_destroy_read_codec()

```
v2f_error_t v2f_file_destroy_read_codec (
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Free all resources allocated when reading the compressor/decompressor pair. Note that this function is only guaranteed to work if the compressor and decompressor were produced by v2f_file_read_codec.

Parameters

compressor	pointer to the compressor to destroy.
decompressor	pointer to the decompressor to destroy

Returns

- V2F_E_NONE : Destruction successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.46.2.6 v2f_file_destroy_read_forest()

```
v2f_error_t v2f_file_destroy_read_forest (
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Free all memory allocated by v2f_file_read_forest. Freeing other coders/decoders may cause a crash.

Parameters

coder	coder to be destroyed.
decoder	decoder to be destroyed.

Returns

- V2F_E_NONE : Destruction successful
- V2F_E_INVALID_PARAMETER : Invalid coder or decoder

5.46.2.7 v2f_file_read_big_endian()

```
v2f_error_t v2f_file_read_big_endian (
            FILE * input_file,
            v2f_sample_t * sample_buffer,
            uint64_t max_sample_count,
            uint8_t bytes_per_sample,
            uint64_t * read_sample_count )
```

Read up to *max_sample_count* samples in big endian format from *input_file*, and store them into *sample_buffer*. The *bytes_per_sample* parameter must be compatible with the definition of the v2f_sample_t type.

Samples are read from *input_file* until either *max_sample_count* samples have been read, or until EOF is found. The destination buffer should be ready to hold up to the maximum requested number of samples.

Note that if the number of samples obtained is less than *max_sample_count*, then V2F_E_UNEXPECTED_END↩ _OF_FILE is returned, and *read_sample_count* is updated with the actual value. However, if the EOF is not aligned with *bytes_per_sample*, V2F_E_IO is returned instead. Based on this, attempting to read from an empty file or at SEEK_END will result in returning V2F_E_UNEXPECTED_END_OF_FILE.

Parameters

input_file	file open for reading.
sample_buffer	buffer with space for at least `max_sample_count` elements.
max_sample_count	maximum number of samples to be read. Cannot be larger than V2F_C_MAX_BLOCK_SIZE.
bytes_per_sample	number of bytes per sample used for reading
read_sample_count	if not NULL, the total number of samples stored in *sample_buffer* is stored there. Otherwise, it is ignored.

Returns

- V2F_E_NONE : All requested samples were successfully read.
- V2F_E_UNEXPECTED_END_OF_FILE : Less than *max_sample_count* samples were copied to the buffer. The *read_sample_count* was updated with the actual count. The end of file occurred with correct alignment.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_IO : An I/O error ocurred, or found EOF not aligned with *bytes_per_sample*,

5.46.2.8 v2f_file_read_codec()

```
v2f_error_t v2f_file_read_codec (
            FILE * input_file,
```

```
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Read a compressor/decompressor pair from *input_file*.

Parameters

input_file	file input for reading
compressor	compressor to be initialized
decompressor	decompressor to be initialized

Returns

- V2F_E_NONE : Read successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.46.2.9 v2f_file_read_forest()

```
v2f_error_t v2f_file_read_forest (
            FILE * input,
            v2f_entropy_coder_t *const coder,
            v2f_entropy_decoder_t *const decoder )
```

Read a file containing the definition of a V2F forest. Produce a coder and decoder pair.

Parameters

input	file containing the definition, produced as defined inv2f_file_write_forest.
coder	pointer to the coder that will be initialized with the read data
decoder	pointer to the decoder that will be initialized with the read data

Returns

- V2F_E_NONE : Coder successfully dumped.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_OUT_OF_MEMORY : Memory could not be initialized for the decoder.
- V2F_E_IO : I/O error saving the coder representation.

5.46.2.10 v2f_file_write_big_endian()

```
v2f_error_t v2f_file_write_big_endian (
            FILE * output_file,
            v2f_sample_t *const sample_buffer,
            uint64_t sample_count,
            uint8_t bytes_per_sample )
```

Write samples to a file, storing them in big endian order.

Parameters

output_file	file open for writing.
sample_buffer	buffer of samples to be writen.
sample_count	number of samples in the buffer.
bytes_per_sample	number of bytes per output

Returns

- V2F_E_NONE : Samples successfully writen
- V2F_E_INVALID_PARAMETER : Invalid input parameters
- V2F_E_IO : Could not write the block

5.46.2.11 v2f_file_write_codec()

```
v2f_error_t v2f_file_write_codec (
            FILE * output_file,
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Write a compressor/decompressor pair to *output_file* with the following format:

- quantizer:
  - mode: 1 byte, must correspond to one of v2f_quantizer_mode_t
  - step size: 4 bytes, unsigned big endian
- decorrelator:
  - mode: 2 bytes, unsigned big endian, must correspond to one of v2f_decorrelator_mode_t
  - max_sample_value: 4 bytes, unsigned big endian. Must be at least as large as the maximum sample value for the V2F forest defined below, if present
- forest_id: 4 bytes, unsigned bit endian. If this field is set to 0, it indicates that a explicit definition of the V2F forest is included afterwards. If set to a larger value v, the (v-1)-th prebuilt forest is used.
- V2F forest: If forest_id != 0, an entropy coder/decoder pair definition, as output by v2f_file_write_forest (variable length).

Parameters

output_file	file open for writing
compressor	initialized compressor of the pair
decompressor	initialized decompressor of the pair

Returns

- V2F_E_NONE : Codec written successfull

- V2F_E_IO : Input/output error
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.46.2.12 v2f_file_write_forest()

```
v2f_error_t v2f_file_write_forest (
            FILE * output,
            v2f_entropy_coder_t const *const coder,
            v2f_entropy_decoder_t const *const decoder,
            uint32_t different_roots )
```

Write the configuration data needed to represent a coder. A decoder can also be extracted from the output data.

Format:

- number of entries: total number of entries, counting those that do not have a word assigned and are not needed by the decoder, 4 bytes, big endian, unsigned,

  See also

  V2F_C_MIN_ENTRY_COUNT,
  V2F_C_MAX_ENTRY_COUNT

- bytes per word: number of bytes used to represent each of the nodes that are included in the V2F dictionary, 1 byte, unsigned,

  See also

  V2F_C_MIN_BYTES_PER_WORD,
  V2F_C_MAX_BYTES_PER_WORD.

- bytes per sample: samples are represented with these many bytes, 1 byte, unsigned,

  See also

  V2F_C_MIN_BYTES_PER_SAMPLE,
  V2F_C_MAX_BYTES_PER_SAMPLE

- max expected value : max expected input sample value, 2 bytes, big endian, unsigned,

  See also

  V2F_C_MAX_SAMPLE_VALUE, must be consistent with `bytes per sample`

- root count : n-1, where n is the number of root nodes included, must be >= 1, 2 bytes BE unsigned,

  See also

  V2F_C_MAX_ROOT_COUNT,
  V2F_C_MAX_CHILD_COUNT.

- For each root:
  - total entry count : total number of entries in this root, 4 bytes, big endian,

See also

V2F_C_MIN_ENTRY_COUNT,
V2F_C_MAX_ENTRY_COUNT.

- number of included entries: total number of entries, excluding those that do not have a word assigned, 4 bytes, big endian, unsigned. Must be consistent with the value of `bytes per word`.
- entries : variable length, `total entry count` elements (included those not included in the decoder dict) ordered by index For each entry in this root:
  - index: 4 bytes, big endian, unsigned. Even though this value can be deduced by the "ordered by index" constraint, this value is maintained to facilitate verification.
  - number of children : 4 bytes, big endian, unsigned
    See also

    V2F_C_MAX_CHILD_COUNT

    If `number of children` $>$ 0, then the following items are also included for this entry of this root:
  - children indices: `number of children` $*$ 4 bytes, ordered by corresponding sample index,

  If `number of children` is not identical to `max expected value` + 1, then node is included in the V2F decoder and has an assigned codeword. In this case, the following fields are present:
  - sample count : number of samples represented by this index, 2 bytes, big endian,
    See also

    V2F_C_MIN_SAMPLE_COUNT,
    V2F_C_MAX_SAMPLE_COUNT
  - samples : `bytes per sample` $*$ `sample count` sample values in `bytes per sample` bytes per sample, big endian, unsigned. Not present if `sample count` is zero.
  - word : bytes corresponding to this included node, `bytes per word` bytes, big endian, unsigned
- number of children of the root node: 4 bytes, big endian, unsigned This number may not exceed the maximum number of possible sample values, but it may be lower than that. If it is, the number of children must be (possible values - i), where i is the index of this root. In this case, children must be assigned for input sample values from i onwards.
- indices of the children of the root node: indices of the roots's entries order by input symbol. For each children:
  - index: 4 bytes, unsigned big endian
  - input symbol value: `bytes per sample` bytes, unsigned big endian

Note that root count can be smaller than `max expected value` In that case, it is to be interpreted that all remaining roots are identical to the last one included.

Parameters

output	file open for writing
coder	coder of the coder/decoder pair.
decoder	decoder of the coder/decoder pair.
different_roots	only the first `different_roots` roots are saved by this function.

Returns

- V2F_E_NONE : Coder successfully dumped.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_IO : I/O error saving the coder representation.

5.46.2.13 v2f_verify_forest()

```
v2f_error_t v2f_verify_forest (
            v2f_entropy_coder_t *const coder,
            v2f_entropy_decoder_t *const decoder )
```

Verify the validity of a coder/decoder pair. Useful when the pair is loaded from a file.

Parameters

coder	pointer to an initialized coder
decoder	pointer to an initialized de coder

Returns

- V2F_E_NONE : Verification successful
- V2F_E_INVALID_PARAMETER : Invalid coder or decoder

5.47 v2f_file.h File Reference

```
#include "v2f.h"
#include "v2f_compressor.h"
#include "v2f_decompressor.h"
```

Functions

- v2f_error_t v2f_file_write_codec (FILE *output_file, v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_read_codec (FILE *input_file, v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_destroy_read_codec (v2f_compressor_t *const compressor, v2f_decompressor_t *const decompressor)
- v2f_error_t v2f_file_write_forest (FILE *output, v2f_entropy_coder_t const *const coder, v2f_entropy_decoder_t const *const decoder, uint32_t different_roots)
- v2f_error_t v2f_file_read_forest (FILE *input, v2f_entropy_coder_t *const coder, v2f_entropy_decoder_t *const decoder)
- v2f_error_t v2f_file_destroy_read_forest (v2f_entropy_coder_t *coder, v2f_entropy_decoder_t *decoder)
- v2f_error_t v2f_verify_forest (v2f_entropy_coder_t *const coder, v2f_entropy_decoder_t *const decoder)
- v2f_error_t v2f_file_read_big_endian (FILE *input_file, v2f_sample_t *sample_buffer, uint64_t max_↩ sample_count, uint8_t bytes_per_sample, uint64_t *read_sample_count)
- v2f_error_t v2f_file_write_big_endian (FILE *output_file, v2f_sample_t *const sample_buffer, uint64_↩ t sample_count, uint8_t bytes_per_sample)

5.47.1 Detailed Description

Interface to handle files.

5.47.2 Function Documentation

5.47.2.1 v2f_file_destroy_read_codec()

```
v2f_error_t v2f_file_destroy_read_codec (
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Free all resources allocated when reading the compressor/decompressor pair. Note that this function is only guaranteed to work if the compressor and decompressor were produced by v2f_file_read_codec.

Parameters

compressor	pointer to the compressor to destroy.
decompressor	pointer to the decompressor to destroy

Returns

- V2F_E_NONE : Destruction successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.47.2.2 v2f_file_destroy_read_forest()

```
v2f_error_t v2f_file_destroy_read_forest (
            v2f_entropy_coder_t * coder,
            v2f_entropy_decoder_t * decoder )
```

Free all memory allocated by v2f_file_read_forest. Freeing other coders/decoders may cause a crash.

Parameters

coder	coder to be destroyed.
decoder	decoder to be destroyed.

Returns

- V2F_E_NONE : Destruction successful
- V2F_E_INVALID_PARAMETER : Invalid coder or decoder

5.47.2.3 v2f_file_read_big_endian()

```
v2f_error_t v2f_file_read_big_endian (
            FILE * input_file,
```

```
v2f_sample_t * sample_buffer,
uint64_t max_sample_count,
uint8_t bytes_per_sample,
uint64_t * read_sample_count )
```

Read up to *max_sample_count* samples in big endian format from *input_file*, and store them into *sample_buffer*. The *bytes_per_sample* parameter must be compatible with the definition of the v2f_sample_t type.

Samples are read from *input_file* until either *max_sample_count* samples have been read, or until EOF is found. The destination buffer should be ready to hold up to the maximum requested number of samples.

Note that if the number of samples obtained is less than *max_sample_count*, then V2F_E_UNEXPECTED_END↩ _OF_FILE is returned, and *read_sample_count* is updated with the actual value. However, if the EOF is not aligned with *bytes_per_sample*, V2F_E_IO is returned instead. Based on this, attempting to read from an empty file or at SEEK_END will result in returning V2F_E_UNEXPECTED_END_OF_FILE.

Parameters

input_file	file open for reading.
sample_buffer	buffer with space for at least `max_sample_count` elements.
max_sample_count	maximum number of samples to be read. Cannot be larger than V2F_C_MAX_BLOCK_SIZE.
bytes_per_sample	number of bytes per sample used for reading
read_sample_count	if not NULL, the total number of samples stored in *sample_buffer* is stored there. Otherwise, it is ignored.

Returns

- V2F_E_NONE : All requested samples were successfully read.
- V2F_E_UNEXPECTED_END_OF_FILE : Less than *max_sample_count* samples were copied to the buffer. The *read_sample_count* was updated with the actual count. The end of file occurred with correct alignment.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_IO : An I/O error ocurred, or found EOF not aligned with *bytes_per_sample*,

5.47.2.4 v2f_file_read_codec()

```
v2f_error_t v2f_file_read_codec (
            FILE * input_file,
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Read a compressor/decompressor pair from *input_file*.

Parameters

input_file	file input for reading
compressor	compressor to be initialized
decompressor	decompressor to be initialized

Returns

- V2F_E_NONE : Read successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.47.2.5 v2f_file_read_forest()

```
v2f_error_t v2f_file_read_forest (
            FILE * input,
            v2f_entropy_coder_t *const coder,
            v2f_entropy_decoder_t *const decoder )
```

Read a file containing the definition of a V2F forest. Produce a coder and decoder pair.

Parameters

input	file containing the definition, produced as defined inv2f_file_write_forest.
coder	pointer to the coder that will be initialized with the read data
decoder	pointer to the decoder that will be initialized with the read data

Returns

- V2F_E_NONE : Coder successfully dumped.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_OUT_OF_MEMORY : Memory could not be initialized for the decoder.
- V2F_E_IO : I/O error saving the coder representation.

5.47.2.6 v2f_file_write_big_endian()

```
v2f_error_t v2f_file_write_big_endian (
            FILE * output_file,
            v2f_sample_t *const sample_buffer,
            uint64_t sample_count,
            uint8_t bytes_per_sample )
```

Write samples to a file, storing them in big endian order.

Parameters

output_file	file open for writing.
sample_buffer	buffer of samples to be writen.
sample_count	number of samples in the buffer.
bytes_per_sample	number of bytes per output

Returns

- V2F_E_NONE : Samples successfully writen
- V2F_E_INVALID_PARAMETER : Invalid input parameters
- V2F_E_IO : Could not write the block

5.47.2.7 v2f_file_write_codec()

```
v2f_error_t v2f_file_write_codec (
            FILE * output_file,
            v2f_compressor_t *const compressor,
            v2f_decompressor_t *const decompressor )
```

Write a compressor/decompressor pair to *output_file* with the following format:

- quantizer:
  - mode: 1 byte, must correspond to one of v2f_quantizer_mode_t
  - step size: 4 bytes, unsigned big endian
- decorrelator:
  - mode: 2 bytes, unsigned big endian, must correspond to one of v2f_decorrelator_mode_t
  - max_sample_value: 4 bytes, unsigned big endian. Must be at least as large as the maximum sample value for the V2F forest defined below, if present
- forest_id: 4 bytes, unsigned bit endian. If this field is set to 0, it indicates that a explicit definition of the V2F forest is included afterwards. If set to a larger value v, the (v-1)-th prebuilt forest is used.
- V2F forest: If forest_id != 0, an entropy coder/decoder pair definition, as output by v2f_file_write_forest (variable length).

Parameters

output_file	file open for writing
compressor	initialized compressor of the pair
decompressor	initialized decompressor of the pair

Returns

- V2F_E_NONE : Codec written successfull
- V2F_E_IO : Input/output error
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.47.2.8 v2f_file_write_forest()

```
v2f_error_t v2f_file_write_forest (
            FILE * output,
```

```
v2f_entropy_coder_t const *const coder,
v2f_entropy_decoder_t const *const decoder,
uint32_t different_roots )
```

Write the configuration data needed to represent a coder. A decoder can also be extracted from the output data.

Format:

- number of entries: total number of entries, counting those that do not have a word assigned and are not needed by the decoder, 4 bytes, big endian, unsigned,

  See also

  V2F_C_MIN_ENTRY_COUNT,
  V2F_C_MAX_ENTRY_COUNT

- bytes per word: number of bytes used to represent each of the nodes that are included in the V2F dictionary, 1 byte, unsigned,

  See also

  V2F_C_MIN_BYTES_PER_WORD,
  V2F_C_MAX_BYTES_PER_WORD.

- bytes per sample: samples are represented with these many bytes, 1 byte, unsigned,

  See also

  V2F_C_MIN_BYTES_PER_SAMPLE,
  V2F_C_MAX_BYTES_PER_SAMPLE

- max expected value : max expected input sample value, 2 bytes, big endian, unsigned,

  See also

  V2F_C_MAX_SAMPLE_VALUE, must be consistent with `bytes per sample`

- root count : n-1, where n is the number of root nodes included, must be >= 1, 2 bytes BE unsigned,

  See also

  V2F_C_MAX_ROOT_COUNT,
  V2F_C_MAX_CHILD_COUNT.

- For each root:
  - total entry count : total number of entries in this root, 4 bytes, big endian,

    See also

    V2F_C_MIN_ENTRY_COUNT,
    V2F_C_MAX_ENTRY_COUNT.

  - number of included entries: total number of entries, excluding those that do not have a word assigned, 4 bytes, big endian, unsigned. Must be consistent with the value of `bytes per word`.
  - entries : variable length, `total entry count` elements (included those not included in the decoder dict) ordered by index For each entry in this root:
    - index: 4 bytes, big endian, unsigned. Even though this value can be deduced by the "ordered by index" constraint, this value is maintained to facilitate verification.
    - number of children : 4 bytes, big endian, unsigned

See also

V2F_C_MAX_CHILD_COUNT

If `number of children` > 0, then the following items are also included for this entry of this root:

* children indices: `number of children` * 4 bytes, ordered by corresponding sample index,

If `number of children` is not identical to `max expected value` + 1, then node is included in the V2F decoder and has an assigned codeword. In this case, the following fields are present:

* sample count : number of samples represented by this index, 2 bytes, big endian,
  See also

  V2F_C_MIN_SAMPLE_COUNT,
  V2F_C_MAX_SAMPLE_COUNT
* samples : `bytes per sample` * `sample count` sample values in `bytes per sample` bytes per sample, big endian, unsigned. Not present if `sample count` is zero.
* word : bytes corresponding to this included node, `bytes per word` bytes, big endian, unsigned

– number of children of the root node: 4 bytes, big endian, unsigned This number may not exceed the maximum number of possible sample values, but it may be lower than that. If it is, the number of children must be (possible values - i), where i is the index of this root. In this case, children must be assigned for input sample values from i onwards.

– indices of the children of the root node: indices of the roots's entries order by input symbol. For each children:

* index: 4 bytes, unsigned big endian
* input symbol value: `bytes per sample` bytes, unsigned big endian

Note that root count can be smaller than `max expected value` In that case, it is to be interpreted that all remaining roots are identical to the last one included.

Parameters

output	file open for writing
coder	coder of the coder/decoder pair.
decoder	decoder of the coder/decoder pair.
different_roots	only the first `different_roots` roots are saved by this function.

Returns

- V2F_E_NONE : Coder successfully dumped.
- V2F_E_INVALID_PARAMETER : Invalid input parameters.
- V2F_E_IO : I/O error saving the coder representation.

5.47.2.9 v2f_verify_forest()

```
v2f_error_t v2f_verify_forest (
            v2f_entropy_coder_t *const coder,
            v2f_entropy_decoder_t *const decoder )
```

Verify the validity of a coder/decoder pair. Useful when the pair is loaded from a file.

Parameters

coder	pointer to an initialized coder
decoder	pointer to an initialized de coder

Returns

- V2F_E_NONE : Verification successful
- V2F_E_INVALID_PARAMETER : Invalid coder or decoder

5.48 v2f_quantizer.c File Reference

```
#include "v2f_quantizer.h"
#include <assert.h>
#include "log.h"
#include "common.h"
```

Functions

- v2f_error_t v2f_quantizer_create (v2f_quantizer_t *quantizer, v2f_quantizer_mode_t mode, v2f_sample_t step_size)
- v2f_error_t v2f_quantizer_quantize (v2f_quantizer_t *const quantizer, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_dequantize (v2f_quantizer_t *const quantizer, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_apply_uniform_division (v2f_sample_t step_size, v2f_sample_t *const input_↩ samples, uint64_t sample_count)
- v2f_error_t v2f_quantize_apply_uniform_shift (v2f_sample_t shift, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_inverse_uniform (v2f_sample_t step_size, v2f_sample_t *const input_samples, uint64_t sample_count)

5.48.1 Detailed Description

Author

Miguel Hernández Cabronero miguel.hernandez@uab.cat

Date

08/03/2021

Tools to apply quantization

5.48.2 Function Documentation

5.48.2.1 v2f_quantize_apply_uniform_shift()

```
v2f_error_t v2f_quantize_apply_uniform_shift (
            v2f_sample_t shift,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply uniform quantization by right-shifting each sample.

Parameters

shift	number of bitplanes to erase.
input_samples	samples to be quantized.
sample_count	number of samples to be quantized.

Returns

- V2F_E_NONE : Creation successfull.
- V2F_E_INVALID_PARAMETER : At least one invalid parameter.

5.48.2.2 v2f_quantizer_apply_uniform_division()

```
v2f_error_t v2f_quantizer_apply_uniform_division (
            v2f_sample_t step_size,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply uniform quantization by dividing each sample

Parameters

step_size	number to be used when dividing
input_samples	samples to be quantized
sample_count	number of samples to be quantized

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.48.2.3 v2f_quantizer_create()

```
v2f_error_t v2f_quantizer_create (
            v2f_quantizer_t * quantizer,
            v2f_quantizer_mode_t mode,
            v2f_sample_t step_size )
```

Initialize a quantizer.

Parameters

quantizer	pointer to the quantizer to initialize.
mode	unique mode id for this quantizer.
step_size	quantization step size. Must be at least 1. If 1, no quantization is performed.

Returns

- V2F_E_NONE : Creation successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.48.2.4 v2f_quantizer_dequantize()

```
v2f_error_t v2f_quantizer_dequantize (
            v2f_quantizer_t *const quantizer,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Dequantize all samples in the block.

Parameters

quantizer	pointer to the quantizer to be used.
input_samples	quantization indices to be dequantized.
sample_count	number of samples to dequantize.

Returns

- V2F_E_NONE : Dequantization successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.48.2.5 v2f_quantizer_inverse_uniform()

```
v2f_error_t v2f_quantizer_inverse_uniform (
            v2f_sample_t step_size,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply inverse uniform quantization for a given step size. The interval midpoint is used.

Parameters

step_size	original step size.
input_samples	buffer of quantization indices to be dequantized.
sample_count	number of indices to dequantizer.

Returns

- V2F_E_NONE : Creation successfull.
- V2F_E_INVALID_PARAMETER : At least one invalid parameter.

5.48.2.6 v2f_quantizer_quantize()

```
v2f_error_t v2f_quantizer_quantize (
            v2f_quantizer_t *const quantizer,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Quantize all samples in the block.

Parameters

quantizer	pointer to the quantizer to be used
input_samples	samples to be quantized (they are modified)
sample_count	number of samples to be quantized

Returns

- V2F_E_NONE : Quantization successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.49 v2f_quantizer.h File Reference

```
#include "v2f.h"
```

Functions

- v2f_error_t v2f_quantizer_create (v2f_quantizer_t *quantizer, v2f_quantizer_mode_t mode, v2f_sample_t step_size)
- v2f_error_t v2f_quantizer_quantize (v2f_quantizer_t *const quantizer, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_dequantize (v2f_quantizer_t *const quantizer, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_apply_uniform_division (v2f_sample_t step_size, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantize_apply_uniform_shift (v2f_sample_t shift, v2f_sample_t *const input_samples, uint64_t sample_count)
- v2f_error_t v2f_quantizer_inverse_uniform (v2f_sample_t step_size, v2f_sample_t *const input_samples, uint64_t sample_count)

5.49.1 Detailed Description

Module that provides quantization tools.

5.49.2 Function Documentation

5.49.2.1 v2f_quantize_apply_uniform_shift()

```
v2f_error_t v2f_quantize_apply_uniform_shift (
            v2f_sample_t shift,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply uniform quantization by right-shifting each sample.

Parameters

shift	number of bitplanes to erase.
input_samples	samples to be quantized.
sample_count	number of samples to be quantized.

Returns

- V2F_E_NONE : Creation successfull.
- V2F_E_INVALID_PARAMETER : At least one invalid parameter.

5.49.2.2 v2f_quantizer_apply_uniform_division()

```
v2f_error_t v2f_quantizer_apply_uniform_division (
            v2f_sample_t step_size,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply uniform quantization by dividing each sample

Parameters

step_size	number to be used when dividing
input_samples	samples to be quantized
sample_count	number of samples to be quantized

Returns

- V2F_E_NONE : Creation successfull
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.49.2.3 v2f_quantizer_create()

```
v2f_error_t v2f_quantizer_create (
            v2f_quantizer_t * quantizer,
```

```
        v2f_quantizer_mode_t mode,
        v2f_sample_t step_size )
```

Initialize a quantizer.

Parameters

quantizer	pointer to the quantizer to initialize.
mode	unique mode id for this quantizer.
step_size	quantization step size. Must be at least 1. If 1, no quantization is performed.

Returns

- V2F_E_NONE : Creation successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.49.2.4 v2f_quantizer_dequantize()

```
v2f_error_t v2f_quantizer_dequantize (
            v2f_quantizer_t *const quantizer,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Dequantize all samples in the block.

Parameters

quantizer	pointer to the quantizer to be used.
input_samples	quantization indices to be dequantized.
sample_count	number of samples to dequantize.

Returns

- V2F_E_NONE : Dequantization successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

5.49.2.5 v2f_quantizer_inverse_uniform()

```
v2f_error_t v2f_quantizer_inverse_uniform (
            v2f_sample_t step_size,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Apply inverse uniform quantization for a given step size. The interval midpoint is used.

Parameters

step_size	original step size.
input_samples	buffer of quantization indices to be dequantized.
sample_count	number of indices to dequantizer.

Returns

- V2F_E_NONE : Creation successfull.
- V2F_E_INVALID_PARAMETER : At least one invalid parameter.

5.49.2.6 v2f_quantizer_quantize()

```
v2f_error_t v2f_quantizer_quantize (
            v2f_quantizer_t *const quantizer,
            v2f_sample_t *const input_samples,
            uint64_t sample_count )
```

Quantize all samples in the block.

Parameters

quantizer	pointer to the quantizer to be used
input_samples	samples to be quantized (they are modified)
sample_count	number of samples to be quantized

Returns

- V2F_E_NONE : Quantization successful
- V2F_E_INVALID_PARAMETER : At least one invalid parameter

Index