基于JPEG图像隐写的红外图像原始数据存储软件

一、.h文件

```h
#pragma once

#include <stdio.h>
#include <iostream>
#include <fstream>
#include <regex>
#include <string>
#include <iterator>
#include <jpeglib.h>

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>

#include "../../lvgl/lvgl.h"
#include "../../lv_drivers/wayland/wayland.h"

cv::Mat image_test();

/**
 * @brief DCIR's Image class, encode image to jpeg with audio, original IR data,
 * @brief and others informations.
 *
 * @note Steganos file after jpeg, use label to split.
 * @note Attached file will be encode to base64.
 */
class DCIR_IMAGE
{
public:
    /* split label = { first, second, last }; total 3 bytes */
    enum split_last_label {
        split_last_label_wav = 0x00,  // wav sound file
        split_last_label_original,    // original IR data
        split_last_label_appendix,    // other sensors info
        split_last_label_nums,        // nums of enum
    };

private:
    using vu8 = std::vector<uint8_t>;
    const uint8_t m_split_first_label = 0xFF;
    const uint8_t m_split_second_label = 0xFF;

    // i.e. JPEG_COM
    vu8 commentLabel = {
        0xFF, 0xFE
    };
    vu8 wavLabel = {
```

```
        m_split_first_label, m_split_second_label, split_last_label_wav
    };
    vu8 originaLabel = {
        m_split_first_label, m_split_second_label,
split_last_label_original
    };
    vu8 appendixLabel = {
        m_split_first_label, m_split_second_label,
split_last_label_appendix
    };

    /* Binary Data, include: jpeg, wav, original, appendix */
    vu8 m_bin;

/* ----- Members ----- */

    /* filepath, name and path */
    std::string m_filepath;
    /* Graphic Mat, open jpeg as cv::Mat */
    cv::Mat m_mat;
    /* Graphic Comment, jpeg's comment */
    std::string m_comment;
    /* Wav Audio */
    vu8 m_wav;
    /* Original IR data */
    vu8 m_original;
    /* Appendix */
    vu8 m_appendix;

/* ----- Debug Function ----- */

    void save_wav();

/* ----- Private Function ----- */

    vu8 read_image_binary(const std::string& filepath);
    void write_vector_binary(const vu8& data, const std::string& filepath);

    vu8 base64_decode(const std::string& base64);
    std::string base64_encode(const std::vector<uint8_t>& data);

    void extract_data();
    std::string extract_comment();
    vu8 extract_wav(const vu8& beginSeq, const vu8& endSeq);
    vu8 extract_original(const vu8& beginSeq, const vu8& endSeq);
    vu8 extract_appendix(const vu8& beginSeq);
    vu8 extract_wrap(vu8::iterator it_begin, vu8::iterator it_end, size_t
beginSeqSize);

    int YUYV_to_JPG(uint8_t* yuvData,
        int imgWidth, int imgHeight,
        const char* fileName, const char* comment);
    vu8 BGR_to_YUYV(const cv::Mat& bgrMat);

public:
    DCIR_IMAGE() = delete;
```

```
    // DCIR_IMAGE(const DCIR_IMAGE&) = delete;
    const DCIR_IMAGE& operator=(const DCIR_IMAGE&) = delete;

/* ----- Constructor ----- */

    DCIR_IMAGE(const std::string& filepath);
    DCIR_IMAGE(
        const std::string& filepath,
        uint8_t * bgrData, size_t bgrLength, int imgWidth, int imgHeight,
        const std::string& comment = std::string{},
        uint8_t * wavData = nullptr, size_t wavLength = 0,
        uint8_t * irData = nullptr, size_t irLength = 0,
        uint8_t * appendix = nullptr, size_t appendixLength = 0);
    ~DCIR_IMAGE();

/* ----- Public Function ----- */

    void save_jpeg(const std::string& filepath);
    void reopen(const std::string& filepath);

/* ----- Get and Set ----- */

    std::string get_filepath();
    cv::Mat get_mat();
    std::string get_comment();
    vu8 get_wav();
    vu8 get_original();
    vu8 get_appendix();
    cv::Mat get_mat_resize(const int& width, const int& height);

    void set_filepath(const std::string& filepath);
    void set_mat(const cv::Mat& mat);
    void set_comment(const std::string& comment);
    void set_wav(const vu8& wav);
    void set_original(const vu8& original);
    void set_appendix(const vu8& appendix);
};
```

一、.cpp文件

```
#include "dcir_image.h"
#include "wav.h"

cv::Mat image_test()
{
    DCIR_IMAGE di("/root/image/output.jpg");

    // std::cout << play_wav() << std::endl;

    // di.set_comment("测试");
    // di.save_jpeg("output.jpg");

    return di.get_mat_resize(100, 100);
}

```

```cpp
/**
 *
================================================================================
=====
 *
================================================================================
=====
 * @par Debug
 *
================================================================================
=====
 *
================================================================================
=====
 */

/**
 * @brief save m_wav to wav file.
 */
void DCIR_IMAGE::save_wav()
{
    std::ofstream outputFile("output.wav", std::ios::binary);
    if (outputFile.is_open())
    {
        outputFile.write(reinterpret_cast<const char*>(m_wav.data()),
m_wav.size());
        outputFile.close();
        std::cout << "Decoded data saved to output.wav" << std::endl;
    }
    else
    {
        std::cout << "Unable to open output.wav for writing" << std::endl;
    }
}

/**
 *
================================================================================
=====
 *
================================================================================
=====
 * @par Private
 *
================================================================================
=====
 *
================================================================================
=====
 */

/**
 * @brief Read image as binary.
 *
 * @param filepath image path.
 * @return uint8_t binary vector.
```

```
 */
std::vector<uint8_t> DCIR_IMAGE::read_image_binary(const std::string&
filepath)
{
    std::ifstream file(filepath, std::ios::binary);

    if (!file) {
        std::cerr << "Failed to open file: " << filepath << std::endl;
        return {};
    }

    // Get the file size
    file.seekg(0, std::ios::end);
    std::streampos fileSize = file.tellg();
    file.seekg(0, std::ios::beg);

    // Read the file into a vector
    std::vector<uint8_t> imageData(fileSize);
    file.read(reinterpret_cast<char*>(imageData.data()), fileSize);

    if (!file) {
        std::cerr << "Failed to read file: " << filepath << std::endl;
        return {};
    }

    file.close();
    return imageData;
}

/**
 * @brief write vector's data as binary.
 *
 * @param data data to be written.
 * @param filepath filename and path.
 */
void DCIR_IMAGE::write_vector_binary(const std::vector<uint8_t>& data,
const std::string& filepath)
{
    std::ofstream file(filepath, std::ios::binary);

    if (!file) {
        std::cerr << "Failed to open file: " << filepath << std::endl;
        return;
    }

    file.write(reinterpret_cast<const char*>(data.data()), data.size());

    if (!file) {
        std::cerr << "Failed to write file: " << filepath << std::endl;
        return;
    }

    file.close();
}

/**
```

```cpp
 * @brief Base64 decode function.
 *
 * @param base64 encoded string.
 * @return decoded string(std::vector).
 */
std::vector<uint8_t> DCIR_IMAGE::base64_decode(const std::string& base64)
{
    static const std::string base64_chars =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

    std::vector<uint8_t> decoded_data;

    // Convert each character in the base64 string to its corresponding 6-bit value
    for (size_t i = 0; i < base64.length(); i += 4)
    {
        uint32_t sextets[4];
        for (size_t j = 0; j < 4; ++j) {
            auto it = std::find(base64_chars.begin(), base64_chars.end(), base64[i + j]);
            if (it != base64_chars.end())
            {
                sextets[j] = std::distance(base64_chars.begin(), it);
            } else
            {
                // Padding character ('=')
                sextets[j] = 0;
            }
        }

        // Convert 4 sextets to 3 bytes
        uint8_t byte1 = (sextets[0] << 2) | (sextets[1] >> 4);
        uint8_t byte2 = (sextets[1] << 4) | (sextets[2] >> 2);
        uint8_t byte3 = (sextets[2] << 6) | sextets[3];

        // Add the decoded bytes to the result
        decoded_data.push_back(byte1);
        if (base64[i + 2] != '=')
            decoded_data.push_back(byte2);
        if (base64[i + 3] != '=')
            decoded_data.push_back(byte3);
    }

    return decoded_data;
}

/**
 * @brief Base64 encode function.
 *
 * @param data vector of bytes to encode.
 * @return encoded string.
 */
std::string DCIR_IMAGE::base64_encode(const std::vector<uint8_t>& data)
{
    static const std::string base64_chars =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
```

```
163
164     std::string encoded_data;
165     size_t data_size = data.size();
166
167     // Iterate over input data in chunks of 3 bytes
168     for (size_t i = 0; i < data_size; i += 3)
169     {
170         // Extract 3 bytes from the input data
171         uint8_t byte1 = data[i];
172         uint8_t byte2 = (i + 1 < data_size) ? data[i + 1] : 0;
173         uint8_t byte3 = (i + 2 < data_size) ? data[i + 2] : 0;
174
175         // Encode the 3 bytes as 4 sextets
176         uint32_t sextet1 = byte1 >> 2;
177         uint32_t sextet2 = ((byte1 & 0x03) << 4) | (byte2 >> 4);
178         uint32_t sextet3 = ((byte2 & 0x0F) << 2) | (byte3 >> 6);
179         uint32_t sextet4 = byte3 & 0x3F;
180
181         // Convert the sextets to base64 characters
182         encoded_data += base64_chars[sextet1];
183         encoded_data += base64_chars[sextet2];
184         encoded_data += (i + 1 < data_size) ? base64_chars[sextet3] : '=';
185         encoded_data += (i + 2 < data_size) ? base64_chars[sextet4] : '=';
186     }
187
188     return encoded_data;
189 }
190
191 /**
192  * @brief Extract(split) binary data from m_bin.
193  */
194 void DCIR_IMAGE::extract_data()
195 {
196     m_comment = extract_comment();
197     // std::cout << m_comment << std::endl;
198     m_wav = extract_wav(wavLabel, originaLabel);
199     extract_original(originaLabel, appendixLabel);
200     extract_appendix(appendixLabel);
201 }
202
203 std::string DCIR_IMAGE::extract_comment()
204 {
205     auto it = std::search(
206         m_bin.begin(), m_bin.end(),
207         commentLabel.begin(), commentLabel.end()
208     );
209
210     if (it != m_bin.end())
211     {
212         // +2 is JPEG_COM(0xFF 0xFE, 2bytes); +4 is comment's length bytes
(2 bytes)
213         std::vector<uint8_t> lengthVec(it + 2, it + 4);
214         size_t length = static_cast<size_t>((lengthVec[0] << 8) |
lengthVec[1]);
215         // std::cout << length << std::endl;
216
```

```
        // +4 is (JPEG_COM + length bytes); -2 is (length bytes)
        return std::string{ it + 4, it + 4 + length - 2};
    }
    else
    {
        // no comment
        return {};
    }
}

/**
 * @brief Extract wav data from m_bin.
 *
 * @param beginSeq wav label, eg. 0xFF 0xFF 0x00
 * @param endSeq original label eg. 0xFF 0xFF 0x01
 * @return wav binary array.
 */
std::vector<uint8_t> DCIR_IMAGE::extract_wav(
    const std::vector<uint8_t>& beginSeq, const std::vector<uint8_t>&
endSeq)
{
    auto it_begin = std::search(
        m_bin.begin(), m_bin.end(),
        beginSeq.begin(), beginSeq.end()
    );
    auto it_end = std::search(
        m_bin.begin(), m_bin.end(),
        endSeq.begin(), endSeq.end()
    );

    return extract_wrap(it_begin, it_end, beginSeq.size());
}

/**
 * @brief Extract original data from m_bin.
 *
 * @param beginSeq original label, eg. 0xFF 0xFF 0x01
 * @param endSeq appendix label eg. 0xFF 0xFF 0x02
 * @return original binary array.
 */
std::vector<uint8_t> DCIR_IMAGE::extract_original(
    const std::vector<uint8_t>& beginSeq, const std::vector<uint8_t>&
endSeq)
{
    auto it_begin = std::search(
        m_bin.begin(), m_bin.end(),
        beginSeq.begin(), beginSeq.end()
    );
    auto it_end = std::search(
        m_bin.begin(), m_bin.end(),
        endSeq.begin(), endSeq.end()
    );

    return extract_wrap(it_begin, it_end, beginSeq.size());
}

```

```cpp
/**
 * @brief Extract appendix data from m_bin.
 *
 * @param beginSeq appendix label, eg. 0xFF 0xFF 0x02
 * @return original appendix array.
 */
std::vector<uint8_t> DCIR_IMAGE::extract_appendix(const
std::vector<uint8_t>& beginSeq)
{
    auto it_begin = std::search(
        m_bin.begin(), m_bin.end(),
        beginSeq.begin(), beginSeq.end()
    );

    return extract_wrap(it_begin, m_bin.end(), beginSeq.size());
}

/**
 * @brief Extract(split) binary data, return m_bin's subsequence.
 *
 * @param it_begin subsequence's begin.
 * @param it_end subsequence's end.
 * @param beginSeqSize begin label's size.
 * @return m_bin's subsequence after base64 decode.
 */
std::vector<uint8_t> DCIR_IMAGE::extract_wrap(
    std::vector<uint8_t>::iterator it_begin,
    std::vector<uint8_t>::iterator it_end,
    size_t beginSeqSize)
{
    if (it_begin != m_bin.end())
    {
        std::ptrdiff_t index = std::distance(m_bin.begin(), it_begin);
        std::cout << "Found at index: " << index << std::endl;

        // Get data after targetSequence
        std::vector<uint8_t> dataAfterEOI(it_begin + beginSeqSize, it_end);

        // base64 decode
        std::string base64_str(dataAfterEOI.begin(), dataAfterEOI.end());
        return base64_decode(base64_str);
    }
    else
    {
        // std::cout << "Not found Sequence" << std::endl;
        return {};
    }
}

/**
 * @brief Save yuv data to jpg.
 *
 * @param yuvData yuyv 422 data.
 * @param imgWidth image width.
 * @param imgHeight image height.
 * @param fileName file name include path.
```

```
 * @param comment jpeg comment.
 *
 * @return success or fail.
 * @retval 0, success.
 * @retval 1, open file fail.
 */
int DCIR_IMAGE::YUYV_to_JPG(
    uint8_t* yuvData,
    int imgWidth, int imgHeight,
    const char* fileName, const char* comment)
{
    int retval = 0;

    /* ===== open file ===== */
    FILE* fp;
    fp = fopen(fileName, "wb");
    if (!fp)
    {
        retval = 1;
        throw std::runtime_error("YUYV_to_JPG(): open file");
        return retval;
    }

    /* ===== jpg init ===== */
    JSAMPROW row_pointer[1];
    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;

    cinfo.err = jpeg_std_error(&jerr);  // init error info first
    jpeg_create_compress(&cinfo);
    jpeg_stdio_dest(&cinfo, fp);

    /* ===== img setting ===== */
    cinfo.image_width = imgWidth;
    cinfo.image_height = imgHeight;
    cinfo.input_components = 3;         // color components for each pixel
    cinfo.in_color_space = JCS_YCbCr;

    jpeg_set_defaults(&cinfo);
    jpeg_set_quality(&cinfo, 75, TRUE);
    jpeg_start_compress(&cinfo, TRUE);

    /* ===== write comment ===== */
    if(comment != nullptr)
        jpeg_write_marker(&cinfo, JPEG_COM, (const JOCTET*)comment,
strlen(comment));

    /* ===== write data ===== */
    uint8_t* buf = (uint8_t*)malloc(sizeof(uint8_t) * imgWidth * 3);
    while (cinfo.next_scanline < cinfo.image_height)
    {
        for (int i = 0; i < cinfo.image_width; i += 2)
        {
            buf[i*3] = yuvData[i*2];
            buf[i*3+1] = yuvData[i*2+1];
            buf[i*3+2] = yuvData[i*2+3];
```

```
            buf[i*3+3] = yuvData[i*2+2];
            buf[i*3+4] = yuvData[i*2+1];
            buf[i*3+5] = yuvData[i*2+3];
        }
        row_pointer[0] = buf;
        yuvData += imgWidth * 2;
        jpeg_write_scanlines(&cinfo, row_pointer, 1);
    }
    jpeg_finish_compress(&cinfo);
    jpeg_destroy_compress(&cinfo);
    free(buf);

out_close_fp:
    if (fp) fclose(fp);

out_return:
    return retval;
}

/**
 * @brief Convert BGR to YUYV.
 *
 * @param bgrMat cv::Mat Which's data is BGR.
 * @return YUYV 422 array.
 */
std::vector<uint8_t> DCIR_IMAGE::BGR_to_YUYV(const cv::Mat& bgrMat)
{
    int width = bgrMat.cols;
    int height = bgrMat.rows;

    // calculate length
    size_t yuyvLength = width * height * 2;

    // create retval
    std::vector<uint8_t> yuyvData(yuyvLength);

    const uint8_t* bgrPtr = bgrMat.data;
    uint8_t* yuyvPtr = yuyvData.data();

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x += 2)
        {
            int bgrIndex1 = y * width * 3 + x * 3;
            int bgrIndex2 = bgrIndex1 + 3;

            // calculate y
            uint8_t y1 = static_cast<uint8_t>(0.299 * bgrPtr[bgrIndex1 + 2] + 0.587 * bgrPtr[bgrIndex1 + 1] + 0.114 * bgrPtr[bgrIndex1]);
            uint8_t y2 = static_cast<uint8_t>(0.299 * bgrPtr[bgrIndex2 + 2] + 0.587 * bgrPtr[bgrIndex2 + 1] + 0.114 * bgrPtr[bgrIndex2]);

            // calculate u & v
            uint8_t u = static_cast<uint8_t>(-0.169 * bgrPtr[bgrIndex1 + 2] - 0.331 * bgrPtr[bgrIndex1 + 1] + 0.5 * bgrPtr[bgrIndex1] + 128);
```

```
            uint8_t v = static_cast<uint8_t>(0.5 * bgrPtr[bgrIndex2 + 2] - 0.419 * bgrPtr[bgrIndex2 + 1] - 0.081 * bgrPtr[bgrIndex2] + 128);

            // write
            yuyvPtr[0] = y1;
            yuyvPtr[1] = u;
            yuyvPtr[2] = y2;
            yuyvPtr[3] = v;

            // move yuyv ptr to next 2 pixel's position
            yuyvPtr += 4;
        }
    }

    return yuyvData;
}

/**
 * ================================================================================
 * ================================================================================
 * @par Public
 * ================================================================================
 * ================================================================================
 */

/**
 * @brief Open existed image file.
 *
 * @param filepath image file path.
 */
DCIR_IMAGE::DCIR_IMAGE(const std::string& filepath)
{
    reopen(filepath);
}

/**
 * @brief Create object with params.
 *
 * @note N means necessary, O means optional.
 *
 * @param filepath          N || filename with path.
 * @param yuvData           N || yuyv422 array to save as jpeg.
 * @param yuvLength         N || yuyv422 length.
 * @param imgWidth          N || image width.
 * @param imgHeight         N || image height.
 *
 * @param comment           O || jpeg comment.
 * @param wavData           O || audo array.
```

```
 * @param wavLength         O || wav length.
 * @param irData            O || original IR sensor output data.
 * @param irLength          O || IR length.
 * @param appendix          O || appendix information.
 * @param appendixLength    O || appendix length.
 */
DCIR_IMAGE::DCIR_IMAGE(
    const std::string& filepath,
    uint8_t * bgrData, size_t bgrLength, int imgWidth, int imgHeight,
    const std::string& comment,
    uint8_t * wavData, size_t wavLength,
    uint8_t * irData, size_t irLength,
    uint8_t * appendix, size_t appendixLength)
{
    if (filepath.empty())
        throw std::runtime_error("filepath is empty.");
    m_filepath = filepath;

    if (bgrData == nullptr)
        throw std::runtime_error("bgrData is nullptr.");
    m_mat = cv::Mat{ imgWidth, imgHeight, CV_8UC3, bgrData };

    if (wavData != nullptr)
    {
        m_wav = std::vector<uint8_t>{ wavData, wavData + wavLength };
    }

    if (irData != nullptr)
    {
        m_original = std::vector<uint8_t>{ irData, irData + irLength };
    }

    if (appendix != nullptr)
    {
        m_appendix = std::vector<uint8_t>{ appendix, appendix +
appendixLength };
    }
}

DCIR_IMAGE::~DCIR_IMAGE()
{
    // do nothing
}

/**
 * @brief save DCIR_IMAGE obj to jpeg file.
 *
 * @param filepath filename and path.
 */
void DCIR_IMAGE::save_jpeg(const std::string& filepath)
{
    /* prepare data */
    auto yuvData = BGR_to_YUYV(m_mat);

    /* save jpeg */
```

```
534     YUYV_to_JPG(yuvData.data(), m_mat.cols, m_mat.rows, filepath.c_str(),
    m_comment.c_str());
535 
536     /* Steganography */
537     std::vector<uint8_t> output = read_image_binary(filepath);
538     if (!m_wav.empty())
539     {
540         output.insert(output.end(), wavLabel.begin(), wavLabel.end());
541         std::string encoded = base64_encode(m_wav);
542         output.insert(output.end(), encoded.begin(), encoded.end());
543     }
544     if(!m_original.empty())
545     {
546         output.insert(output.end(), originaLabel.begin(),
    originaLabel.end());
547         std::string encoded = base64_encode(m_original);
548         output.insert(output.end(), encoded.begin(), encoded.end());
549     }
550     if(!m_appendix.empty())
551     {
552         output.insert(output.end(), appendixLabel.begin(),
    appendixLabel.end());
553         std::string encoded = base64_encode(m_appendix);
554         output.insert(output.end(), encoded.begin(), encoded.end());
555     }
556 
557     write_vector_binary(output, filepath);
558 }
559 
560 /**
561  * @brief reset all element, i.e. open a new image,
562  * @brief instead of destroy and create new DCIR_IMAGE obj.
563  *
564  * @param filepath filename and path.
565  */
566 void DCIR_IMAGE::reopen(const std::string& filepath)
567 {
568     /* Step 1 : Filepath */
569     m_filepath = filepath;
570 
571     /* Step 2 : Mat */
572     m_mat = cv::imread(m_filepath);
573 
574     /* Step 3 : Comment, Wav, Original, Appendix */
575     m_bin = read_image_binary(m_filepath);
576     extract_data();
577 }
578 
579 /**
580  *
    ================================================================================
    =====
581  *
    ================================================================================
    =====
582  * @par Get and Set
```

```
583  *
     ================================================================================
     =====
584  *
     ================================================================================
     =====
585  */
586
587  std::string DCIR_IMAGE::get_filepath()
588  {
589      return m_filepath;
590  }
591
592  cv::Mat DCIR_IMAGE::get_mat()
593  {
594      return m_mat;
595  }
596
597  std::string DCIR_IMAGE::get_comment()
598  {
599      return m_comment;
600  }
601
602  std::vector<uint8_t> DCIR_IMAGE::get_wav()
603  {
604      return m_wav;
605  }
606  std::vector<uint8_t> DCIR_IMAGE::get_original()
607  {
608      return m_original;
609  }
610  std::vector<uint8_t> DCIR_IMAGE::get_appendix()
611  {
612      return m_appendix;
613  }
614
615  /**
616   * @brief get a resized cv::mat obj, which is assigned by m_mat.
617   *
618   * @param width width of resize.
619   * @param height height of resize.
620   * @return resized m_mat.
621   */
622  cv::Mat DCIR_IMAGE::get_mat_resize(const int& width, const int& height)
623  {
624      // return mat
625      cv::Mat image;
626      cv::Size targetSize(width, height);
627
628      cv::resize(m_mat, image, targetSize, 0, 0, cv::INTER_LINEAR);
629
630      return image;
631  }
632
633  void DCIR_IMAGE::set_filepath(const std::string& filepath)
634  {
```

```
635     m_filepath = filepath;
636 }
637
638 void DCIR_IMAGE::set_mat(const cv::Mat& mat)
639 {
640     m_mat = mat;
641 }
642
643 void DCIR_IMAGE::set_comment(const std::string& comment)
644 {
645     m_comment = comment;
646 }
647
648 void DCIR_IMAGE::set_wav(const std::vector<uint8_t>& wav)
649 {
650     m_wav = wav;
651 }
652
653 void DCIR_IMAGE::set_original(const std::vector<uint8_t>& original)
654 {
655     m_original = original;
656 }
657
658 void DCIR_IMAGE::set_appendix(const std::vector<uint8_t>& appendix)
659 {
660     m_appendix = appendix;
661 }
```