User Help file for EncodedProjectApp

1. About the program:

This program can decode text messages, which was encoded into bmp images. After that it sends the text message to a server with a specified id. You can also use this program in other modes. You can learn more about these at the 3rd paragraph. Please use bmp files for the correct software usage.

2. How to make the app runnable?

- To use the program you will need gcc to compile it to your computer.
- You can use the following gcc switches:
    - ➔ '-fopenmp' to enable Multi-thread support.
    - ➔ '-o' to set the compiled file's name.
    - ➔ '-w' to disable warning messages during
- Do the following things to make the app runnable, if you are not familiar with it.
    - ➔ Open a Terminal in the folder where you have the EncodedProject.c file.

      (For this a dedicated folder is recommended.)
    - ➔ Use one of the following commands:
        - "gcc EncodedProject.c -o EncodedProjectApp -w"

          if you DON'T want multi-thread support
        - "gcc EncodedProject.c -o EncodedProjectApp -w -fopenmp"

          if you want multi-thread support
    - ➔ Use the following command to run the program:
        - "./EncodedProjectApp"
- You can also use arguments to run the program not just in the standard mode. See more information in the 3rd paragraph.

3. How to use the program?

- The program will not stop if you press Ctrl+C during the data decoding process.
  Please keep this in mind!
- The program also tries to avoid infinite loops with timeouts.
- You can run the program in different modes. These are:
- ➔ Normal mode: Without argument
    - In this mode the program decodes an encoded bmp image. The program will open a command line file explorer at your home folder.
    - How the file explorer works?

If the input is a Folder, then it opens it, and writes out the folder's content. The browsing starts at the user's home folder. The browsing ends if the user gives a file as an input. If the file or folder name is not found, it stays in the current directory. You can go to a parent folder with the '..' "command".

- ➔ Normal mode: With one argument - '[path_to_the_encoded_image]'

  In this mode the program decodes an encoded bmp image. The program reads the only parameter as the chosen file's path. This path must be either relative path from the app's run directory or an absolute path. Make sure to give the right path for a successful decoding.
- ➔ Help mode: Argument: '--help'

  Gives you some instructions about the usable arguments.
- ➔ About mode: Argument: '--version'

  Gives you some information about the software's actual version and its creator.
- ➔ Test mode: Argument: '--test'

  This mode is for testing and presentation purposes. Encode an 'abc' string into 3 randomly generated pixels and show their decoded and encoded content. Also sends this test string to the server with the correct ID.
- ➔ POST mode: Arguments: '--send [your_ID] [a_single_word_message]'
  - This mode is for testing purposes. For example, to test if we can reach the server or not.
  - We send a single word message to the server. This is given by the 3rd argument.
  - We send the HTML POST message with your chosen ID, what you have to set in the second argument.

4. The meaning of return values:

1. The software's run finished without an error.
2. The software's memory allocation was unsuccessful.
3. The message what the software tried to send to the server was too big.
4. The software run into an error during the socket creation.
5. The software could not connect to the server.
6. The software run into error while sending the message to the server.
7. The software could not receive the server's message.
8. The software does not receive the correct verification message.
9. The image's decoding process did not complete in 1 second, and this generated a timeout error.
10. The arguments what the user tried to use is unsupported by the software.

5. What are the used functions doing in 'TheProgram.h', and how to use them?

- encode_pixel method: Encodes one char into one bmp pixel.
  - ➔ Encoding logic: encodes a single char's 1st 1/3 into the 1st, 2nd into the 2nd and 3rd into the 3rd color of the pixel (always using the last 3 bits).
  - ➔ Parameters:
    - caracter (in): The character what we want to encode into the pixel.
    - pixelArray (out): A pointer to the start of the pixel's data we want to encode.
- TestArray method: Creates a random test array of pixels with an already encoded 'abc' message.
  - ➔ Parameters:
    - NumCh (out): An int pointer where the method saves the number of encoded characters.
    - return: An int pointer of the generated test array.
- Unwrap method: Decodes an already encoded pixel array by the same logic what is definied in the encode_pixel method.
  - ➔ Parameters:
    - Pbuff (out): The decoded image data buffer's start pointer.
    - NumCh (out): The number of encoded characters.
    - @return: The decoded text message's start pointer.
- ReadPixels method: Opens a bmp file's data and gets its pixel array and the # of encoded caracters.
  - ➔ Parameters:
    - f (in): The opened file stream of the encoded bmp image.
    - NumCh (out): A pointer, where the function saves the # of encoded caracters.
    - return: The pixel array of the bmp image.
- BrowseForOpen method: A method, what creates a Terminal File browser. If the input is a Folder, then it opens it, and writes out the folder's content. The browsing starts at the user's home folder. And ends if the user gives a file as an input. If the file or folder name is not found, it stays in the current directory. You can go to a parent folder with the '..' "command".
  - ➔ Parameters:
    - return: A read-only file stream of the chosen file.
- Post method: Sends a HTML POST message, with the neptunID and message to the correct IP adress.
  - ➔ Parameters:
    - neptunID (in): The user's neptunID. You will find your message with the help of this ID.
    - message (in): The message what you want to send to the server.
    - NumCh (in): The length of the message.

    - return: Returns 0, if successful.
      Returns an error code in the range of 3-8 with the corresponding meaning
    - In the case of error, this also writes out an error message into the standard error output.

- WhatToDo method: This function catches the SIGINT and SIGALRM signals. In the case of a SIGINT signal the process starts a child process (with forking), which reminds the user, that the program disabled the key combination. After that it kills the process immediately. In the case of a SIGALRM signal the process creates a timeout error message and exits with a '9' error code.
  - ➔ Parameters:
    - sig (in): The catched signal's identifier flag.
- char_to_binary_str method: An extra char to binary text converter for displaying and debug purposes.
  - ➔ Parameters:
    - a (in): The character what we want to convert to binary string.

Created by Matthew Vagner