Object-Oriented Programming VL

Laborprotokoll

Beispiel 2

Günther Neuwirth, Matr. Nr.0626638
e0626638@student.tuwien.ac.at
Manuel Mausz, Matr. Nr.0728348
manuel-tu@mausz.at

Wien, am 3. Mai 2009

Inhaltsverzeichnis

1 Aufgabenstellung - Beispiel 2

Beispielangaben zu
OOP (Objekt-Orientierte Programmierung)
SS 2009

Beispiel 2

Relevante Themen:

- Untertyp-Beziehungen
- Zusicherungen
- Fehlerbehandlung

Design und Implementierung:
Schreiben Sie in C++ ein skriptgesteuertes Bildbearbeitungsprogramm imgsynth2. Wie der Name schon andeutet, handelt es sich hier um eine Erweiterung des Programms imgsynth der ersten Beispielangabe.

Die Synopsis des Programms ist unverändert: imgsynth2 -i <scriptfile>
Die Änderungen betreffen eine Erweiterung der zu unterstützenden Befehle und zu unterstützenden Bildformate.

So sollen zusätzlich zu den Befehlen read(),fillrect() und write() noch folgende Befehle zusätzlich unterstützt werden:

invert() … Invertiert das Bild Invertieren heißt, für jedes Pixel und allen Farbkomponenten (R,G,B) der aktuelle Wert vom Maximalwert subtrahiert wird:

$$val_{neu} = MAXVAL - val_{alt}$$

brightness(faktor) … Ändert die Helligkeit der Pixel. Für jedes Pixel wird jede Farbkomponente folgendermaßen aktualisiert:

$$val_{neu} = \min (MAXVAL, val_{alt} * faktor)$$

mirror_y() … Spiegelt das Bild um die Y-Achse (horizontales Spiegeln)

mirror_x() … Spiegelt das Bild um die X-Achse (vertikales Spiegeln)

Weiters sollen zusätzliche Bildformate unterstützt werden.
Zum einen soll die Unterstützung des „Windows Bitmap“ Formates der ersten Beispielangabe (Typ „BMP“) um ein weiteres Pixelformat erweitert werden. Mit dem Eintrag Pixelformat=16 soll das Format B5G5R5 unterstützt werden, welches fünf Bits für die blaue, fünf Bits für die grüne und fünf Bits für die rote Farbkomponente benutzt.

Damit man auf die 16 Bit kommt, wird die Farbkomponente Blau mit sechs Bits gespeichert, jedoch nur die niederwertigsten fünf Bits werden verwendet.

Weiters soll noch das Bildformat XPM unterstützt werden, welches ein ASCII-Textfile mit ANSI C-Syntax darstellt (dieses Format wurde ursprünglich für das X-Window System entwickelt und erlaubt die einfache Benutzung von Bildern in C Programmen). Obwohl dieses Bildformat in Textform gespeichert ist, ist das Einlesen nicht allzu schwer. Man muss aber beachten, dass dieses Bildformat im Gegensatz zu den beiden BMP Formaten mit Farbpaletten arbeitet. Mit Farbpaletten werden Farbwerte als Paare von Identifier und Farbwert definiert und die einzelnen Bildpunkte definieren nur den Identifier als Farbindex. Im Folgenden ist ein Beispiel gezeigt, wo die Definition der Farbpaletten Fett hervorgehoben wurde:

```
[0] man_yellow.xpm
/* XPM */
static char *man_yellow_xpm[]={
"9 17 2 1",
". c #001010",
"# c #ffdc3a",
".........",
".........",
"...###...",
"...###...",
"...###...",
"....#....",
"..#####..",
".#.###.#.",
".#.###.#.",
".#.###.#.",
".#.###.#.",
"...#.#...",
"...#.#...",
"...#.#...",
"...#.#...",
".........",
"........."};
```

Oberhalb der Farbpalettendefinition befindet sich eine Headerzeile mit den Werten X, Y, #C und #c, wobei X die Länge der Pixelzeilen definiert (9 im Beispiel), Y die Anzahl der Pixelzeilen definiert (17 im Beispiel), #C die Anzahl der Farben definiert (2 im Beispiel) und #c die Anzahl der Textzeichen pro Farbpixel definiert.

Der Einfachheit halber sei angenommen, dass ANSI C Kommentare nur am Zeilenanfang beginnen dürfen. Weiters braucht nur die Variante unterstützt werden, wo die Farben direkt als Binärwerte definiert werden (Varianten, wo Farben mit symbolischen Namen definiert werden, braucht daher nicht unterstützt werden). Weitere Informationen zum XPM Bildformat findet man beispielsweise im Internet unter folgender Adresse:

http://en.wikipedia.org/wiki/X_PixMap

Für dieses Beispiel ist es notwendig, dass man das Klassendesign gegenüber dem ersten Beispiel refaktorisiert. Die Klasse CBitmap soll nun eine abstrakte Interfaceklasse sein,

von welcher Klassen für die jeweiligen Dateiformate abgeleitet werden. Ebenso soll es für die Klasse CPixelFormat gemacht werden, welche nun abstrakte Interfaceklasse für die Klassen der unterschiedlichen Pixelcodierungen ist. In dem Beispiel werden zumindest drei Pixelcodierungen benötigt: RGB16, RGB24 und Indexed8. Das Pixelformat meint eine Farbkodierung mit Paletten und 8 Bit breiten Farbindizes.

Die in der Spezifikation nicht genauer ausgeführten Teile können selbst sinnvoll ausgestaltet werden, wobei jedoch die folgenden Anforderungen einzuhalten sind.

Bonusaufgabe (freiwillig, ermöglicht 5 Bonuspunkte)

- Unterstützung weiterer einfachen Dateiformaten (z.B. TGA)
-
- Skriptbefehle in dynamisch ladbaren Modulen implementiert (Plugin-Mechanismus für beliebige Erweiterungen).

Anforderungen:
Kommentieren Sie den Code, um das Verstehen des Codes zu vereinfachen. Zusätzlich soll zu Beginn jeder Klasse eine Klassenbeschreibung stehen. Am Beginn jeder Datei soll außerdem Name, MNr, KZ der Gruppenmitglieder sowie die Beispielnummer stehen. Ein nicht kommentierter Code wird negativ bewertet!

Ausnahmen (Exceptions) müssen immer abgefangen werden, sofern möglich. Überlegen Sie sich dabei aber auch, welche Ausnahmen zu einer Programmbeendigung führen müssen und in welchen Fällen es ausreicht, eine Warnung bzw. Fehlermeldung auszugeben. Bei jeder Ausnahmebehandlung ist auf jeden Fall eine Fehlermeldung auf die Fehlerausgabe (cerr) auszugeben. Als Orientierung seien einige mögliche Quellen für Ausnahmen aufgezählt: Speichermangel, inkorrekter Input, Schreiben auf Datei fehlgeschlagen, etc.

Schreiben Sie klare Zusicherungen an den passenden Stellen im Programm. Dort, wo man Zusicherungen kompakt als Code formulieren kann, wird empfohlen, die Zusicherungen zusätzlich als „assert()“ hinzuschreiben, damit sie auch vom Compiler geprüft werden können. Mit der Compileroption „-DNDEBUG“ kann man den fertig ausgetesteten Code generieren sodass „assert()“ keinen Overhead mehr verursacht.

Benutzen Sie zur Programmentwicklung auch ein Makefile, welches zumindest die Targets „clean“, „all“ und „run“ enthält. Mittels „make run“ soll das Programm direkt aufgerufen werden können (mit sinnvollen Argumenten versehen).

Es sind die Programmierrichtlinien der LVA einzuhalten.

2 Beispiel 2

2.1 Design

Abbildung 1 zeigt das Klassendiagramm der Aufgabe.

Refaktorisierung:

Notwendigerweise und wie verlangt wurde die Klasse CBitmap zu einer abstrakten Interfaceklasse umgewandelt. Diese stellt nun die Oberklasse der zu unterstützenden Dateiformate dar und enthält verschieden virtuelle Methoden, die von ihren Unterklassen zu implementieren sind. Dazu gehören, die schon aus Beispiel 1 bekannten Methoden read() und write(), welche das Einlesen und Schreiben der Bilddaten realisieren. Hinzugekommen sind nun verschiedene "getter"-Methoden, wie z.B. getHeight() oder getWidth(), die vom direkten Zugriff auf allgemeine Bildinformationen abstrahieren.
Die gemäß Aufgabenstellungen verlangten Befehle zur Bildbearbeiten, wurden in abstrakten Interfaceklasse CBitmap als "protected" generisch für alle Bildformate implementiert. Mit dieser Vorgehensweise war es möglich, die zu unterstützenden Dateiformate auf ihre Eigenheiten zu beschränken, diese zu realisieren und allgemeinen Code wieder zu verwenden. Unterklassen sind CWindowsBitmap aus Aufgabenstellung 1 und für das neu hinzugekommene Format Pixmap die Klasse CPixmap.

Die abstrakte Interfaceklasse CPixelFormat wurde aus Beispiel 1 übernommen und enthält nun neben getPixel() und setPixel() zusätzliche virtuelle Methoden, die von den Unterklassen zu implementieren sind und Informationen zu den von ihnen zu verarbeitenden Pixelcodierung bereitstellen. Weiters wurde die Datenstruktur RGBPIXEL eingeführt, welche zur Übergabe der Farbwerte der einzelnen Pixel dient. Unterklassen sind CPixelformat_BRG24 aus Beispiel 1 sowie die neuen Formate CPixelformat_BRG55 und CPixelformat_Indexed8.

Asserts:

Beim Einlesen und Bearbeiten von Bilddaten wurden Zusicherungen deklariert, welche die Größe und Existenz der Bilddaten sicherstellen.

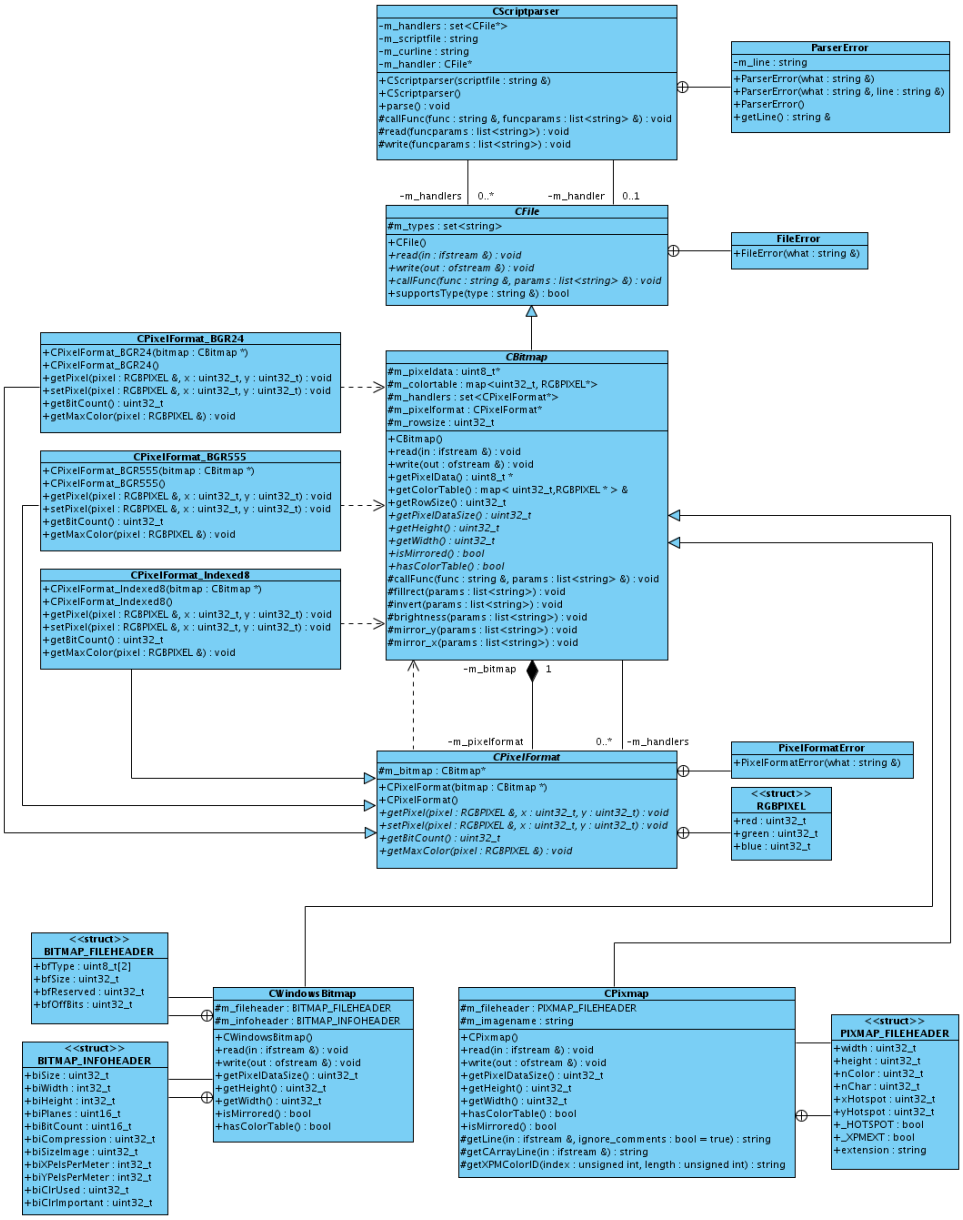

Abbildung 1: Klassendiagramm 1

2.2 Verwaltung der Ressourcen

Die Vorgehensweise bei der Ressourcenverwaltung wurde größtenteils von Aufgabe 1 übernommen. Neu hinzugekommen sind Funktionalitäten zur Verwaltung von Farbtabellen. Dabei wurde ebenfalls ein generisches Konzept gewählt, welches Unterschiede in der Farbindizierung ausgleichen soll. Erwähnt sei, dass in der Implementierung der Klasse CPixmap vorerst nur Farbwerte in hexadezimaler Form sowie der Pixmap eigene Darstellungsmodus “c” unterstützt werden. Beim Einlesen der Pixeldaten, welche die Indizes der zugeordneten Farbwert darstellen, werden diese mit internen, allgemeineren Indizes der Form 0,1,2 .. n ausgetauscht. Die Farbtabelle wird in einer “std::map”, welche im abstrakten Interface CBitmap deklariert ist und die internen Indizes als Schlüssel erwartet, abgelegt. Farbwerte werden dabei in der in Abschnitt 2.1 erwähnten Datenstruktur RGBPIXEL, für die dynamisch Speicher alloziert werden muss, organisiert. Die Freigabe dieser Ressourcen erfolgt wieder im Destruktor von CBitmap.

2.3 Fehlerbehandlung

Die Fehlerbehandlung erfolgt nach wie vor über die Übersetzung der Exceptions CPixelFormat::PixelFormatError, CFile::FileError, CScriptParser::ParserError und dem “try-catch” Block im Hauptprogramm. Die Header und Pixeldaten werden beim Einlesen rudimentär, aber gemäß der Spezifikation, auf Korrektheit überprüft. Im Fehlerfall wird eine Exception geworfen, eine Fehlerbeschreibung über stderr ausgegeben und das Programm beendet.

2.4 Implementierung

Im Folgenden werden einige Implementierungauszüge der geforderten Funktionen gezeigt. Siehe auch Punkt 2.1 und Abbildung 1 sowie Punkt 4.

Im Falle eines Bildes mit Farbtabelle ist es nur notwendig die Farbwerte in der Tabelle zu ändern.

Die Farbtabelle ist in folgender Form in der abstrakten Interfaceklasse CBitmap deklariert.

```
std::map<uint32_t, CPixelFormat::RGBPIXEL *> m_colortable;
```

Beim Schreiben der Bilddaten für das Format Pixmap werden die Pixel und Indizes der Farbtabelle wieder in korrekte Identifier zurückkonvertiert.

```
#define PIXMAP_COLORCHARS ".#abcdefghijklmnopqrstuvwxyzABCD" \
                          "EFGHIJKLMNOPQRSTUVWXYZ0123456789"
```

invert(): Beispiel für die Invertierung eines Farbwertes in einer Rastergrafik.

```
pixel.red   = max.red   - pixel.red;
```

brightness(params): Beispiel für die Veränderung der Helligkeit eines Bildpunktes.

```
pixel.red   = min(max.red,   static_cast<uint32_t>(pixel.red   * factor));
```

3 Projektverlauf

3.1 Probleme und Fallstricke

Die Schwierigkeit bestand darin, einen Weg zu finden, Bildformate mit Farbtabellen gleichermaßen zu behandeln, wie jene, die die Farbwerte direkt im Bildspeicher halten. Durch die virtuelle Methode const bool hasColorTable() in der abstrakten Interfaceklasse CBitmap ist es möglich die Abarbeitung von Bildoperationen auf den Bilddaten entsprechend zu delegieren.

3.2 Arbeitsaufwand

Entwicklungsschritt / Meilenstein	Arbeitsaufwand in Stunden
Erstes Design	2 Stunden
Refaktorisierung	6 Stunden
Implementierung (und Anpassung des Designs)	5 Tag
Dokumentation (Doxygen) und Überprüfung aller Anforderungen gemäß der Programmierrichtlinien	3 Stunden
Erstellung des Protokolls	3 Stunden

4 Listings

4.1 imgsynth2.cpp

```
/**
 * @module imgsynth2
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  imgsynth2 reads a scriptfile given as commandline option
 *         and executes all known function inside.
 *         On error (e.g. unknown function) the program will terminate
 * @date   17.04.2009
 * @par Exercise
 *      2
 */

#include <iostream>
#include <boost/program_options.hpp>
#include "cscriptparser.h"

using namespace std;
namespace po = boost::program_options;

/**
 * @func   main
 * @brief  program entry point
 * @param  argc  standard parameter of main
 * @param  argv  standard parameter of main
 * @return 0 on success, not 0 otherwise
 * @globalvars none
 * @exception  none
 * @conditions none
 *
 * setup commandline options, parse them and pass scriptfile to scriptparser
 * instance. On error print error message to stderr.
 * Unknown commandline options will print a usage message.
 */
int main(int argc, char* argv[])
{
  string me(argv[0]);

  /* define commandline options */
  po::options_description desc("Allowed options");
  desc.add_options()
    ("help,h", "this help message")
    ("input,i", po::value<string>(), "input scriptfile");

  /* parse commandline options */
  po::variables_map vm;
  try
  {
    po::store(po::parse_command_line(argc, argv, desc), vm);
    po::notify(vm);
  }
  catch(po::error& ex)
  {
    cerr << "Error: " << ex.what() << endl;
  }

  /* print usage upon request or missing params */
  if (vm.count("help") || !vm.count("input"))
  {
```

```
    cout << "Usage: " << me << " -i <scriptfile>" << endl;
    cout << desc << endl;
    return 0;
  }

  CScriptparser parser(vm["input"].as<string>());
  try
  {
    parser.parse();
  }
  catch(CScriptparser::ParserError& ex)
  {
    cerr << me << ": Error while processing scriptfile: " << ex.what() << endl;
    if (!ex.getLine().empty())
      cerr << "Scriptline: '" << ex.getLine() << "'" << endl;
    return 1;
  }
  catch(exception& ex)
  {
    cerr << me << ": Unexpected exception: " << ex.what() << endl;
    return 1;
  }

  return 0;
}

/* vim: set et sw=2 ts=2: */
```

4.2 cscriptparser.h

```
/**
 * @module cscriptparser
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  class for parsing simple scriptfiles
 * @date   17.04.2009
 */

#ifndef CSCRIPTPARSER_H
#define CSCRIPTPARSER_H

#include <stdexcept>
#include <string>
#include <list>
#include <set>
#include "cfile.h"

/**
 * @class CScriptparser
 *
 * Parses a simple line based scriptfile with some limitations:
 * first function (starting a block) must be a read-command,
 * last must be a write-command (ending this block).
 *
 * read- and write-commands have hard coded parameters, number#1 being a filetype.
 * Classes handling certain filetypes must be of type CFile.
 * Custom functions will be passed to CFile::callFunc().
 *
 * On error ParserError will be thrown.
 */
class CScriptparser
{
  public:
    /**
     * @class ParserError
     * @brief Exception thrown by CScriptparser
     */
    class ParserError : public std::invalid_argument {
      public:
        /**
         * @method ParserError
         * @brief  Default exception ctor
         * @param  what message to pass along
         * @return -
         * @globalvars none
         * @exception  none
         * @conditions none
         */
        ParserError(const std::string& what)
          : std::invalid_argument(what), m_line("")
        {}

        /**
         * @method ParserError
         * @brief  Custom exception ctor
         * @param  what message to pass along
         * @param  line scriptline which is currently being parsed
         * @return -
         * @globalvars none
         * @exception  none
         * @conditions none
         */
```

```
    ParserError(const std::string& what, const std::string& line)
      : std::invalid_argument(what), m_line(line)
    {}

    /**
     * @method ~ParserError
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  not allowed
     * @conditions none
     */
    ~ParserError() throw()
    {}

    /**
     * @method getLine
     * @brief  returns reference to currently parsed scriptline (if set)
     * @return reference to currently parsed scriptline (maybe empty string)
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const std::string &getLine()
    {
      return m_line;
    }

  private:
    /* members*/
    std::string m_line;
};

/**
 * @method CScriptparser
 * @brief  Default ctor
 * @param  scriptfile filename of script to parse
 * @return -
 * @globalvars none
 * @exception  bad_alloc
 * @conditions none
 */
CScriptparser(const std::string& scriptfile);

/**
 * @method ~CScriptparser
 * @brief  Default dtor
 * @param  -
 * @return -
 * @globalvars none
 * @exception  none
 * @conditions none
 */
~CScriptparser();

/**
 * @method parse
 * @brief  Start parsing the scriptfile
 * @param  -
 * @return -
 * @globalvars none
 * @exception  ParserError
 * @conditions none
```

```
     */
    void parse();

  protected:
    /**
     * @method callFunc
     * @brief  Delegates the function and its parameters to the correct
     *         method (internal or handler)
     * @param  func       function name
     * @param  funcparams function parameters as list
     * @return -
     * @globalvars none
     * @exception  ParserError
     * @conditions none
     */
    void callFunc(const std::string& func, const std::list<std::string>& funcparams
        );

    /**
     * @method read
     * @brief  Handles/wrappes read-command. according to the filetype the
     *         read-method of the corresponding handler will be called inside.
     * @param  funcparams function parameters as list
     * @return -
     * @globalvars none
     * @exception  ParserError
     * @conditions none
     *
     * Scriptfile syntax: read(<FILETYPE>, <FILENAME>)
     */
    void read(std::list<std::string> funcparams);

    /**
     * @method write
     * @brief  Handles/wrappes write-command. according to the filetype the
     *         write-method of the corresponding handler will be called inside.
     * @param  funcparams function parameters as list
     * @return -
     * @globalvars none
     * @exception  ParserError
     * @conditions none
     *
     * Scriptfile syntax: write(<FILETYPE>, <FILENAME>)
     */
    void write(std::list<std::string> funcparams);

  private:
    /* members */
    std::set<CFile *> m_handlers;
    std::string m_scriptfile;
    std::string m_curline;
    CFile *m_handler;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.3 cscriptparser.cpp

```
/**
 * @module cscriptparser
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  class for parsing simple scriptfiles
 * @date   17.04.2009
 */

#include <fstream>
#include <boost/tokenizer.hpp>
#include <boost/algorithm/string.hpp>
#include "cscriptparser.h"
#include "cwindowsbitmap.h"
#include "cpixmap.h"

using namespace std;
using namespace boost;

CScriptparser::CScriptparser(const std::string& scriptfile)
  : m_scriptfile(scriptfile), m_handler(NULL)
{
  /* add our handlers */
  m_handlers.insert(new CWindowsBitmap);
  m_handlers.insert(new CPixmap);
}

/*-----------------------------------------------------------------------------*/

CScriptparser::~CScriptparser()
{
  /* delete image handlers */
  set<CFile *>::iterator it;
  for (it = m_handlers.begin(); it != m_handlers.end(); it++)
    delete *it;
  m_handler = NULL;
}

/*-----------------------------------------------------------------------------*/

void CScriptparser::parse()
{
  /* open and read file */
  ifstream file(m_scriptfile.c_str(), ios::in);
  if (!file)
    throw ParserError("Unable to open scriptfile '" + m_scriptfile + "'.");

  while (!file.eof() && file.good())
  {
    /* read file pre line */
    getline(file, m_curline);
    if (m_curline.empty())
      continue;

    trim(m_curline);

    /* ignore comments */
    if (m_curline.find_first_of('#') == 0)
      continue;

    /* line has no function call */
    size_t pos1 = m_curline.find_first_of('(');
    size_t pos2 = m_curline.find_last_of(')');
```

```
    if (pos1 == string::npos || pos2 == string::npos)
      throw ParserError("Invalid syntax. Not a function", m_curline);

    /* first parse function name and tokenize all parameters */
    string func = m_curline.substr(0, pos1);
    string params = m_curline.substr(pos1 + 1, pos2 - pos1 - 1);
    list<string> funcparams;
    tokenizer< char_separator<char> > tokens(params, char_separator<char>(","));
    /* BOOST_FOREACH isn't available on OOP-servers... */
    for (tokenizer< char_separator<char> >::iterator it = tokens.begin();
        it != tokens.end();
        it++)
    {
      string tok(*it);
      trim(tok);
      if (tok.find_first_of(' ') != string::npos)
      {
        if (tok.find_first_of('"') == string::npos)
          throw ParserError("Invalid syntax", m_curline);
      }
      trim_if(tok, is_any_of("\""));
      funcparams.push_back(tok);
    }

    /* then call the corresponding function */
    callFunc(func, funcparams);
  }

  file.close();
}

/*-----------------------------------------------------------------------------*/

void CScriptparser::callFunc(const std::string& func, const std::list<std::string>&
     funcparams)
{
  if (func.empty())
    throw ParserError("Function name is empty.", m_curline);

  if (func == "read")
    read(funcparams);
  else if (func == "write")
    write(funcparams);
  else
  {
    if (m_handler == NULL)
      throw ParserError("No image is being processed.", m_curline);

    /* call function from handler */
    try
    {
      m_handler->callFunc(func, funcparams);
    }
    catch(CFile::FileError& ex)
    {
      throw ParserError(ex.what(), m_curline);
    }
  }
}

/*-----------------------------------------------------------------------------*/

void CScriptparser::read(std::list<std::string> funcparams)
{
```

```
  /* check prerequirements */
  if (funcparams.size() != 2)
    throw ParserError("Invalid number of function parameters (must be 2).",
        m_curline);
  if (m_handler != NULL)
    throw ParserError("An image is already being processed. Unable to open another.
        ", m_curline);

  string type = funcparams.front();
  to_upper(type);
  funcparams.pop_front();
  string filename = funcparams.front();

  /* fetch image handler supporting requested filetype */
  m_handler = NULL;
  set<CFile *>::iterator it;
  for (it = m_handlers.begin(); it != m_handlers.end(); it++)
  {
    if ((*it)->supportsType(type))
    {
      m_handler = *it;
      break;
    }
  }
  if (m_handler == NULL)
    throw ParserError("Unknown filetype.", m_curline);

  /* open file in binary mode */
  ifstream file(filename.c_str(), ios::in | ios::binary);
  if (!file)
    throw ParserError("Unable to read file.", m_curline);

  /* let handlers read() parse the file */
  try
  {
    m_handler->read(file);
    if (!file.good())
      throw ParserError("Error while reading image file.", m_curline);
    file.close();
  }
  catch(CFile::FileError& ex)
  {
    file.close();
    throw ParserError(ex.what(), m_curline);
  }
}

/*-------------------------------------------------------------------------*/

void CScriptparser::write(std::list<std::string> funcparams)
{
  /* check prerequirements */
  if (funcparams.size() != 2)
    throw ParserError("Invalid number of function parameters (must be 2).",
        m_curline);
  if (m_handler == NULL)
    throw ParserError("No image is being processed.", m_curline);

  string type = funcparams.front();
  to_upper(type);
  funcparams.pop_front();
  string filename = funcparams.front();

  /* do we have an image handler supporting the filetype? */
```

```
  if (!m_handler->supportsType(type))
    throw ParserError("Unknown filetype.", m_curline);

  /* open file in binary mode */
  ofstream file(filename.c_str(), ios::out | ios::binary);
  if (!file)
    throw ParserError("Unable to open file.", m_curline);

  /* let handlers write() parse the file */
  try
  {
    m_handler->write(file);
    if (!file.good())
      throw ParserError("Error while writing image file.", m_curline);
    file.close();
    m_handler = NULL;
  }
  catch(CFile::FileError& ex)
  {
    file.close();
    m_handler = NULL;
    throw ParserError(ex.what(), m_curline);
  }
}

/* vim: set et sw=2 ts=2: */
```

4.4 cfile.h

```
/**
 * @module cfile
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Abstract class for handling files.
 *         Needed for generic use in CScriptparser.
 * @date   17.04.2009
 */

#ifndef CFILE_H
#define CFILE_H

#include <string>
#include <set>
#include <list>
#include <fstream>
#include <stdexcept>

/**
 * @class CFile
 * @brief Abstract class for handling files. Needed for generic use in
 *        CScriptparser.
 *
 * In order for CScriptparser to determine which instance of CFile supports
 * which filetype, every implemententation need to insert their filetypes to
 * the member m_types in their constructor.
 *
 * On error throw FileError.
 */
class CFile
{
  public:
    /**
     * @class FileError
     * @brief Exception thrown by implemententations of CFile
     */
    class FileError : public std::invalid_argument {
      public:
        /**
         * @method FileError
         * @brief  Default exception ctor
         * @param  what message to pass along
         * @return -
         * @globalvars none
         * @exception  none
         * @conditions none
         */
        FileError(const std::string& what)
          : std::invalid_argument(what)
        {}
    };

    /**
     * @method ~CFile
     * @brief  Default dtor (virtual)
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual ~CFile()
```

```
    {};

    /**
     * @method read
     * @brief  Pure virtual method (interface). Should read data from filestream.
     * @param  in filestream to read data from
     * @return -
     * @globalvars none
     * @exception  FileError
     * @conditions none
     */
    virtual void read(std::ifstream& in) = 0;

    /**
     * @method write
     * @brief  Pure virtual method (interface). Should write data to filestream.
     * @param  out      filestream to write data to
     * @return -
     * @globalvars none
     * @exception  FileError
     * @conditions none
     */
    virtual void write(std::ofstream& out) = 0;

    /**
     * @method callFunc
     * @brief  Pure virtual method (interface). Should delegate the function
     *         and its parameters to the correct internal method.
     * @param  func   function name
     * @param  params function parameters as list
     * @return -
     * @globalvars none
     * @exception  FileError
     * @conditions none
     */
    virtual void callFunc(const std::string& func, const std::list<std::string>&
        params) = 0;

    /**
     * @method supportsType
     * @brief  Check if filetype is supported by this implementation.
     * @param  type filetype
     * @return true if filetype is supported. false otherwise
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    bool supportsType(const std::string& type)
    {
      return (m_types.find(type) == m_types.end()) ? false : true;
    }

  protected:
    /* members */
    /** set of filetypes suppported by this implementation */
    std::set<std::string> m_types;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.5 cbitmap.h

```
/**
 * @module cbitmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Abstract implementation of CFile handling Bitmaps.
 * @date   17.04.2009
 */

#ifndef CBITMAP_H
#define CBITMAP_H

#include <stdint.h>
#include <map>
#include "cfile.h"
#include "cpixelformat.h"

/**
 * @class CBitmap
 * @brief Implementation of CFile handling Bitmaps.
 *
 * In order to support operations on bitmaps with different color bitcounts
 * different implementations of CPixelFormat are used. These classes are
 * allowed to modify the bitmap headers and pixelbuffer directly.
 *
 * On error CFile::FileError is thrown.
 */
class CBitmap : public CFile
{
  public:
    /**
     * @method CBitmap
     * @brief  Default ctor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CBitmap()
      : m_pixeldata(NULL), m_pixelformat(NULL), m_rowsize(0)
    {}


    /**
     * @method ~CBitmap
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual ~CBitmap();

    /**
     * @method read
     * @brief  Reads Windows Bitmap from filestream.
     *         On error an exception is thrown.
     * @param  in filestream to read data from
     * @return -
     * @globalvars none
     * @exception CFile::FileError
```

```
     * @exception bad_alloc
     * @conditions none
     */
    virtual void read(std::ifstream& in) = 0;

    /**
     * @method write
     * @brief  Writes Windows Bitmap to filestream.
     * @param  out      filestream to read data from
     * @return -
     * @globalvars none
     * @exception  FileError
     * @exception  bad_alloc
     * @conditions none
     */
    virtual void write(std::ofstream& out) = 0;

    /**
     * @method getPixelData
     * @brief  Returns pointer to pixelbuffer
     * @param  -
     * @return pointer to pixelbuffer
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    uint8_t *getPixelData()
    {
      return m_pixeldata;
    }

    /**
     * @method getColorTable
     * @brief  Returns reference to colortable
     * @param  -
     * @return reference to colortable
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    std::map<uint32_t, CPixelFormat::RGBPIXEL *>& getColorTable()
    {
      return m_colortable;
    }

    /**
     * @method getRowSize
     * @brief  Returns number of bytes of one row
     * @param  -
     * @return number of bytes of one row
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    uint32_t getRowSize()
    {
      return m_rowsize;
    }

    /**
     * @method getPixelDataSize
     * @brief  Return size of pixelbuffer
     * @param  -
     * @return size of pixelbuffer
```

```
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual const uint32_t getPixelDataSize() = 0;

    /**
     * @method getHeight
     * @brief  Return height of bitmap in pixel
     * @param  -
     * @return height of bitmap in pixel
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual const uint32_t getHeight() = 0;

    /**
     * @method getWidth
     * @brief  Return width of bitmap in pixel
     * @param  -
     * @return width of bitmap in pixel
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual const uint32_t getWidth() = 0;

    /**
     * @method isMirrored
     * @brief  Windows Bitmaps can be stored upside down
     * @param  -
     * @return true if bitmap is stored upside down. false otherwise
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual const bool isMirrored() = 0;

    /**
     * @method hasColorTable
     * @brief  Check if bitmap has a colortable
     *         (we don't support this yet for windows bitmaps)
     * @param  -
     * @return true if bitmap has a colortable. false otherwise
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual const bool hasColorTable() = 0;

  protected:
    /**
     * @method callFunc
     * @brief  Delegates the function and its parameters to the correct
     *         internal method
     * @param  func   function name
     * @param  params function parameters as list
     * @return -
     * @globalvars none
     * @exception  ParserError
     * @conditions none
     */
    void callFunc(const std::string& func, const std::list<std::string>& params);
```

```
/**
 * @method fillrect
 * @brief  Fills rectangle in image starting on position x, y
 *         width size width, height and color red, green, blue.
 * @param  params function parameters as list
 * @return -
 * @globalvars none
 * @exception  FileError
 * @conditions none
 *
 * Scriptfile syntax: fillrect(x, y, width, height, red, green, blue)
 */
void fillrect(std::list<std::string> params);

/**
 * @method invert
 * @brief  Invert image
 * @param  params function parameters as list
 * @return -
 * @globalvars none
 * @exception  FileError
 * @conditions none
 *
 * Scriptfile syntax: invert()
 */
void invert(std::list<std::string> params);

/**
 * @method brightness
 * @brief  Increase/decrease brightness of image
 * @param  params function parameters as list
 * @return -
 * @globalvars none
 * @exception  FileError
 * @conditions none
 *
 * Scriptfile syntax: brightness(factor)
 */
void brightness(std::list<std::string> params);

/**
 * @method mirror_y
 * @brief  Mirror image around the y-axis
 * @param  params function parameters as list
 * @return -
 * @globalvars none
 * @exception  FileError
 * @conditions none
 *
 * Scriptfile syntax: mirror_y()
 */
void mirror_y(std::list<std::string> params);

/**
 * @method mirror_x
 * @brief  Mirror image around the x-axis
 * @param  params function parameters as list
 * @return -
 * @globalvars none
 * @exception  FileError
 * @conditions none
 *
 * Scriptfile syntax: mirror_y()
```

```
     */
    void mirror_x(std::list<std::string> params);

    /* members */
    /** pointer to pixelbuffer */
    uint8_t *m_pixeldata;
    /** colortable map */
    std::map<uint32_t, CPixelFormat::RGBPIXEL *> m_colortable;
    /** set of supported PixelFormat handlers */
    std::set<CPixelFormat *> m_handlers;
    /** pointer to CPixelFormat implementation */
    CPixelFormat *m_pixelformat;
    /** number of bytes of one row in the image */
    uint32_t m_rowsize;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.6 cbitmap.cpp

```
/**
 * @module cbitmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Abstract implementation of CFile handling Bitmaps.
 * @date   17.04.2009
 */

#include <algorithm>
#include <boost/lexical_cast.hpp>
#include <boost/numeric/conversion/cast.hpp>
#include <assert.h>
#include "cbitmap.h"

using namespace std;

/*-----------------------------------------------------------------------------*/

CBitmap::~CBitmap()
{
  /* delete pixeldata */
  if (m_pixeldata != NULL)
    delete[] m_pixeldata;
  m_pixeldata = NULL;

  /* delete pixelformat handlers */
  set<CPixelFormat *>::iterator it;
  for (it = m_handlers.begin(); it != m_handlers.end(); it++)
    delete *it;
  m_pixelformat = NULL;

  /* delete colortable content */
  map<uint32_t, CPixelFormat::RGBPIXEL *>::iterator it2;
  for(it2 = m_colortable.begin(); it2 != m_colortable.end(); it2++)
    delete (*it2).second;
}

/*-----------------------------------------------------------------------------*/

void CBitmap::callFunc(const std::string& func, const std::list<std::string>&
    params)
{
  if (func.empty())
    throw FileError("Function name is empty.");

  if (func == "fillrect")
    fillrect(params);
  else if (func == "brightness")
    brightness(params);
  else if (func == "mirror_x")
    mirror_x(params);
  else if (func == "mirror_y")
    mirror_y(params);
  else if (func == "invert")
    invert(params);
  else
    throw FileError("Unknown function '" + func + "'.");
}

/*-----------------------------------------------------------------------------*/

void CBitmap::fillrect(std::list<std::string> params)
```

```
{
  /* check prerequirements */
  if (params.size() != 7)
    throw FileError("Invalid number of function parameters (must be 7).");

  /* do nothing if no pixel exists */
  if (m_pixeldata == NULL || m_pixelformat == NULL)
    return;

  assert(getHeight() > 0);
  assert(getWidth() > 0);
  assert(getPixelDataSize() > 0);

  /* convert parameters */
  uint32_t pparams[7];
  int i = 0;
  try
  {
    for(i = 0; i < 7; i++)
    {
      pparams[i] = boost::lexical_cast<uint32_t>(params.front());
      params.pop_front();
    }
  }
  catch(boost::bad_lexical_cast& ex)
  {
    throw FileError("Invalid parameter (" + params.front() + ").");
  }

  /* check parameter values are in range */
  if (pparams[0] < 0 || pparams[0] > getWidth()
   || pparams[1] < 0 || pparams[1] > getHeight())
    throw FileError("At least one x/y-parameter is out of range.");

  /* check parameter values are in range */
  CPixelFormat::RGBPIXEL pixel;
  m_pixelformat->getMaxColor(pixel);
  if (pparams[4] < 0 || pparams[4] > pixel.red
   || pparams[5] < 0 || pparams[5] > pixel.green
   || pparams[6] < 0 || pparams[6] > pixel.blue)
    throw FileError("At least one pixel color parameter is out of range.");

  if (pparams[2] < 0 || pparams[2] + pparams[0] > getWidth()
   || pparams[3] < 0 || pparams[3] + pparams[1] > getHeight())
    throw FileError("At least one w/h-parameter is out of range.");

  /* new pixel data */
  pixel.red   = pparams[4];
  pixel.green = pparams[5];
  pixel.blue  = pparams[6];

  /* call setPixel for every pixel in the rectangel */
  /* NOTE: maybe use std::fill() here? */
  for(uint32_t i = pparams[0]; i < pparams[2] + pparams[0]; i++)
  {
    for(uint32_t j = pparams[1]; j < pparams[3] + pparams[1]; j++)
    {
      try
      {
        m_pixelformat->setPixel(pixel, i, j);
      }
      catch(CPixelFormat::PixelFormatError& ex)
      {
        stringstream errstr;
```

```
        errstr << "Can't set pixel (pos=[" << i << "," << j << "] col=["
          << pparams[4] << "," << pparams[5] << "," << pparams[6] << "]): "
          << ex.what();
        throw FileError(errstr.str());
      }
    }
  }
}

/*-----------------------------------------------------------------------------*/

void CBitmap::invert(std::list<std::string> params)
{
  /* check prerequirements */
  if (params.size() != 0)
    throw FileError("Invalid number of function parameters (must be 0).");

  /* do nothing if no pixel exists */
  if (m_pixeldata == NULL || m_pixelformat == NULL)
    return;

  assert(getHeight() > 0);
  assert(getWidth() > 0);
  assert(getPixelDataSize() > 0);

  CPixelFormat::RGBPIXEL pixel;
  CPixelFormat::RGBPIXEL max;
  m_pixelformat->getMaxColor(max);
  if (hasColorTable())
  {
    /* invert every entry in the colortable */
    map<uint32_t, CPixelFormat::RGBPIXEL *>::iterator it;
    for (it = m_colortable.begin(); it != m_colortable.end(); it++)
    {
      (*it).second->red   = max.red   - (*it).second->red;
      (*it).second->green = max.green - (*it).second->green;
      (*it).second->blue  = max.blue  - (*it).second->blue;
    }
  }
  else
  {
    /* invert per pixel */
    for(uint32_t y = 0; y < getHeight(); y++)
    {
      for(uint32_t x = 0; x < getWidth(); x++)
      {
        try
        {
          m_pixelformat->getPixel(pixel, x, y);
          pixel.red   = max.red   - pixel.red;
          pixel.green = max.green - pixel.green;
          pixel.blue  = max.blue  - pixel.blue;
          m_pixelformat->setPixel(pixel, x, y);
        }
        catch(CPixelFormat::PixelFormatError& ex)
        {
          stringstream errstr;
          errstr << "Can't invert pixel (pos=[" << x << "," << y << "]): "
                 << ex.what();
          throw FileError(errstr.str());
        }
      }
    }
  }
```

```
}

/*----------------------------------------------------------------------------*/

void CBitmap::brightness(std::list<std::string> params)
{
  /* check prerequirements */
  if (params.size() != 1)
    throw FileError("Invalid number of function parameters (must be 1).");

  /* do nothing if no pixel exists */
  if (m_pixeldata == NULL || m_pixelformat == NULL)
    return;

  /* convert parameters */
  float factor;
  try
  {
    factor = boost::lexical_cast<float>(params.front());
    params.pop_front();
  }
  catch(boost::bad_lexical_cast& ex)
  {
    throw FileError("Invalid parameter (" + params.front() + ").");
  }

  /* negative factor doesn't make sense */
  if (factor < 0)
    throw FileError("Brightness parameter must be positive.");

  CPixelFormat::RGBPIXEL pixel;
  CPixelFormat::RGBPIXEL max;
  m_pixelformat->getMaxColor(max);
  if (hasColorTable())
  {
    /* change every entry in the colortable */
    map<uint32_t, CPixelFormat::RGBPIXEL *>::iterator it;
    for (it = m_colortable.begin(); it != m_colortable.end(); it++)
    {
      (*it).second->red   = min(max.red,   static_cast<uint32_t>((*it).second->red
          * factor));
      (*it).second->green = min(max.green, static_cast<uint32_t>((*it).second->
          green * factor));
      (*it).second->blue  = min(max.blue,  static_cast<uint32_t>((*it).second->blue
          * factor));
    }
  }
  else
  {
    /* change per pixel */
    for(uint32_t y = 0; y < getHeight(); y++)
    {
      for(uint32_t x = 0; x < getWidth(); x++)
      {
        try
        {
          m_pixelformat->getPixel(pixel, x, y);
          pixel.red   = min(max.red,   static_cast<uint32_t>(pixel.red   * factor))
              ;
          pixel.green = min(max.green, static_cast<uint32_t>(pixel.green * factor))
              ;
          pixel.blue  = min(max.blue,  static_cast<uint32_t>(pixel.blue  * factor))
              ;
          m_pixelformat->setPixel(pixel, x, y);
```

```
        }
        catch(CPixelFormat::PixelFormatError& ex)
        {
          stringstream errstr;
          errstr << "Can't invert pixel (pos=[" << x << "," << y << "]): "
                 << ex.what();
          throw FileError(errstr.str());
        }
      }
    }
  }
}

/*----------------------------------------------------------------------------*/

void CBitmap::mirror_y(std::list<std::string> params)
{
  /* check prerequirements */
  if (params.size() != 0)
    throw FileError("Invalid number of function parameters (must be 0).");

  /* do nothing if no pixel exists */
  if (m_pixeldata == NULL || m_pixelformat == NULL)
    return;

  assert(m_rowsize > 0);
  assert(getHeight() > 0);
  assert(getWidth() > 0);
  assert(getPixelDataSize() > 0);

  uint8_t *buf = new uint8_t[m_rowsize];
  for(uint32_t i = 0; i < getHeight()/2; i++)
  {
    uint32_t j = getHeight() - i - 1;
    uint32_t offset  = i * m_rowsize;
    uint32_t backset = j * m_rowsize;

    /* boundary check */
    if (offset + m_rowsize > getPixelDataSize()
     || backset + m_rowsize > getPixelDataSize())
      throw FileError("Mirrored pixel position is out of range.");

    /* mirroring, backup lower data first */
    copy(m_pixeldata + backset, m_pixeldata + backset + m_rowsize, buf);
    copy(m_pixeldata + offset,  m_pixeldata + offset + m_rowsize, m_pixeldata +
        backset);
    copy(buf, buf + m_rowsize, m_pixeldata + offset);
  }
  delete[] buf;
}

/*----------------------------------------------------------------------------*/

void CBitmap::mirror_x(std::list<std::string> params)
{
  /* check prerequirements */
  if (params.size() != 0)
    throw FileError("Invalid number of function parameters (must be 0).");

  /* do nothing if no pixel exists */
  if (m_pixeldata == NULL || m_pixelformat == NULL)
    return;

  /* calc pixelwidth */
```

```
  unsigned int pixelwidth = (hasColorTable()) ? sizeof(uint32_t) : m_pixelformat->
      getBitCount()/8;

  assert(m_rowsize > 0);
  assert(getHeight() > 0);
  assert(getWidth() > 0);
  assert(getPixelDataSize() > 0);

  uint8_t *buf = new uint8_t[pixelwidth];
  for(uint32_t i = 0; i < getHeight(); i++)
  {
    uint32_t offset = i * m_rowsize;

    for(uint32_t j = 0; j <= getWidth()/2; j++)
    {
      uint32_t poffset  = offset + j * pixelwidth;
      uint32_t pbackset = offset + getWidth() * pixelwidth - j * pixelwidth;

      /* boundary check */
      if (pbackset > getPixelDataSize())
        throw FileError("Mirrored␣pixel␣position␣is␣out␣of␣range.");

      /* mirroring, backup right data first */
      copy(m_pixeldata + pbackset - pixelwidth, m_pixeldata + pbackset, buf);
      copy(m_pixeldata + poffset,  m_pixeldata + poffset + pixelwidth, m_pixeldata
          + pbackset - pixelwidth);
      copy(buf, buf + pixelwidth, m_pixeldata + poffset);
    }
  }
  delete[] buf;
}

/* vim: set et sw=2 ts=2: */
```

4.7 cwindowsbitmap.h

```
/**
 * @module cwindowsbitmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CBitmap handling Windows Bitmaps.
 * @date   17.04.2009
 */

#ifndef CWINDOWSBITMAP_H
#define CWINDOWSBITMAP_H

#include <stdint.h>
#include "cbitmap.h"

/**
 * @class CWindowsBitmap
 * @brief Implementation of CBitmap handling Windows Bitmaps.
 *
 * On error CFile::FileError is thrown.
 */
class CWindowsBitmap : public CBitmap
{
  public:
    /**
     * @method CWindowsBitmap
     * @brief  Default ctor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CWindowsBitmap();

    /**
     * @method ~CWindowsBitmap
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CWindowsBitmap()
    {}

    /**
     * @method read
     * @brief  Reads Windows Bitmap from filestream.
     *         On error an exception is thrown.
     * @param  in filestream to read data from
     * @return -
     * @globalvars none
     * @exception CFile::FileError
     * @exception bad_alloc
     * @conditions none
     */
    void read(std::ifstream& in);

    /**
     * @method write
     * @brief  Writes Windows Bitmap to filestream.
```

```
     * @param  out       filestream to read data from
     * @return -
     * @globalvars none
     * @exception  FileError
     * @exception  bad_alloc
     * @conditions none
     */
    void write(std::ofstream& out);

#ifdef DEBUG
    /**
     * @method dump
     * @brief  Dumps the Windows Bitmap file headers to ostream
     * @param  out output stream
     * @return -
     * @globalvars
     * @exception
     * @conditions
     */
    void dump(std::ostream& out);
#endif

    /**
     * @method getPixelDataSize
     * @brief  Return size of pixelbuffer
     * @param  -
     * @return size of pixelbuffer
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const uint32_t getPixelDataSize()
    {
      return m_infoheader.biSizeImage;
    }

    /**
     * @method getHeight
     * @brief  Return height of bitmap in pixel
     * @param  -
     * @return height of bitmap in pixel
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const uint32_t getHeight()
    {
      /* width and height can be negativ */
      return static_cast<uint32_t>(abs(m_infoheader.biHeight));
    }

    /**
     * @method getWidth
     * @brief  Return width of bitmap in pixel
     * @param  -
     * @return width of bitmap in pixel
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const uint32_t getWidth()
    {
      /* width and height can be negativ */
      return static_cast<uint32_t>(abs(m_infoheader.biWidth));
```

```
    }

    /**
     * @method isMirrored
     * @brief  Windows Bitmaps can be stored upside down
     * @param  -
     * @return true if bitmap is stored upside down. false otherwise
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const bool isMirrored()
    {
      /* if height is positive the y-coordinates are mirrored */
      return (m_infoheader.biHeight > 0) ? true : false;
    }

    /**
     * @method hasColorTable
     * @brief  Check if bitmap has a colortable
     *         (we don't support this yet for windows bitmaps)
     * @param  -
     * @return true if bitmap has a colortable. false otherwise
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const bool hasColorTable()
    {
      return false;
    }

  protected:
    /**
     * @brief Windows Bitmap File Header structure
     */
#pragma pack(push,1)
    typedef struct
    {
      /** the magic number used to identify the BMP file */
      uint8_t  bfType[2];
      /** the size of the BMP file in bytes */
      uint32_t bfSize;
      /** reserved */
      uint32_t bfReserved;
      /** the offset of the byte where the bitmap data can be found */
      uint32_t bfOffBits;
    } BITMAP_FILEHEADER;
#pragma pack(pop)

    /**
     * @brief Windows Bitmap Info Header structure
     */
#pragma pack(push,1)
    typedef struct
    {
      /** the size of this header (40 bytes) */
      uint32_t biSize;
      /** the bitmap width in pixels (signed integer) */
      int32_t  biWidth;
      /** the bitmap height in pixels (signed integer) */
      int32_t  biHeight;
      /** the number of color planes being used. Must be set to 1 */
      uint16_t biPlanes;
```

```
        /** the number of bits per pixel, which is the color depth of the image */
        uint16_t biBitCount;
        /** the compression method being used */
        uint32_t biCompression;
        /** the image size */
        uint32_t biSizeImage;
        /** the horizontal resolution of the image (pixel per meter) */
        int32_t  biXPelsPerMeter;
        /** the vertical resolution of the image (pixel per meter) */
        int32_t  biYPelsPerMeter;
        /** the number of colors in the color palette, or 0 to default to 2^n */
        uint32_t biClrUsed;
        /** the number of important colors used, or 0 when every color is
         * important; generally ignored. */
        uint32_t biClrImportant;
    } BITMAP_INFOHEADER;
#pragma pack(pop)

    /* members */
    /** fileheader */
    BITMAP_FILEHEADER m_fileheader;
    /** infoheader */
    BITMAP_INFOHEADER m_infoheader;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.8 cwindowsbitmap.cpp

```
/**
 * @module cwindowsbitmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CBitmap handling Windows Bitmaps.
 * @date   17.04.2009
 */

#include <boost/lexical_cast.hpp>
#include <boost/numeric/conversion/cast.hpp>
#include <assert.h>
#ifdef DEBUG
# include <iostream>
#endif
#include "cwindowsbitmap.h"
#include "cpixelformat_bgr24.h"
#include "cpixelformat_bgr555.h"

using namespace std;

CWindowsBitmap::CWindowsBitmap()
{
  m_types.insert("BMP");

  /* add our handlers */
  m_handlers.insert(new CPixelFormat_BGR24(this));
  m_handlers.insert(new CPixelFormat_BGR555(this));
}

/*-----------------------------------------------------------------------------*/

void CWindowsBitmap::read(std::ifstream& in)
{
  /* read and check file header */
  in.read(reinterpret_cast<char *>(&m_fileheader), sizeof(m_fileheader));

  if (m_fileheader.bfType[0] != 'B' || m_fileheader.bfType[1] != 'M')
    throw FileError("Imagefile␣has␣invalid␣Bitmap␣header.");
  /* bfSize is unreliable (http://de.wikipedia.org/wiki/Windows_Bitmap) */
  if (m_fileheader.bfSize < 0)
    throw FileError("Bitmap␣filesize␣is␣less␣than␣zero?");

  /* read and check info header */
  in.read(reinterpret_cast<char *>(&m_infoheader), sizeof(m_infoheader));

  if (m_infoheader.biSize != 40)
    throw FileError("Bitmap␣info␣header␣size␣is␣invalid.");
  if (m_infoheader.biPlanes != 1)
    throw FileError("Bitmap␣color␣planes␣is␣not␣set␣to␣1.");
  if (m_infoheader.biCompression != 0)
    throw FileError("Bitmap␣compression␣is␣set␣but␣not␣supported.");
  if (m_infoheader.biSizeImage < 0)
    throw FileError("Bitmap␣image␣size␣is␣less␣than␣zero?");
  if (m_infoheader.biClrUsed != 0 || m_infoheader.biClrImportant != 0)
    throw FileError("Bitmap␣colortable␣is␣used␣but␣not␣supported.");

  /* read pixel data using separate class */
  if (m_infoheader.biSizeImage > 0)
  {
    if (m_pixeldata != NULL)
      delete[] m_pixeldata;
    m_pixeldata = new uint8_t[m_infoheader.biSizeImage];
```

```
    assert(m_fileheader.bfOffBits > 0);
    in.seekg(m_fileheader.bfOffBits);
    in.read(reinterpret_cast<char *>(m_pixeldata), m_infoheader.biSizeImage);
  }

  /* get pixelformat instance */
  m_pixelformat = NULL;
  set<CPixelFormat *>::iterator it;
  for (it = m_handlers.begin(); it != m_handlers.end(); it++)
  {
    if (m_infoheader.biBitCount == (*it)->getBitCount())
    {
      m_pixelformat = *it;
      break;
    }
  }
  if (m_pixelformat == NULL)
    throw FileError("Bitmap bitcount is not supported.");

  /* calc rowsize - boundary is 32 */
  m_rowsize = 4 * static_cast<uint32_t>(
    ((m_pixelformat->getBitCount() * m_infoheader.biWidth) + 31) / 32
  );
}

/*-------------------------------------------------------------------------*/

void CWindowsBitmap::write(std::ofstream& out)
{
  /* set header values */
  m_fileheader.bfSize = m_infoheader.biSizeImage + sizeof(m_infoheader) + sizeof(
      m_fileheader);

  /* write file header */
  out.write(reinterpret_cast<char *>(&m_fileheader), sizeof(m_fileheader));

  /* write info header */
  out.write(reinterpret_cast<char *>(&m_infoheader), sizeof(m_infoheader));

  /* write pixel data */
  if (m_pixeldata != NULL)
    out.write(reinterpret_cast<char *>(m_pixeldata), m_infoheader.biSizeImage);
}

/*-------------------------------------------------------------------------*/

#ifdef DEBUG
void CWindowsBitmap::dump(std::ostream& out)
{
  out
    << "Bitmap File Header:" << endl
    << "  bfType="      << m_fileheader.bfType[0] << m_fileheader.bfType[1]
    << ", bfSize="      << m_fileheader.bfSize
    << ", bfReserved=" << m_fileheader.bfReserved
    << ", bfOffBits="  << m_fileheader.bfOffBits
    << endl;

  out
    << "Bitmap Info Header:" << endl
    << "  biSize="    << m_infoheader.biSize
    << ", biWidth="  << m_infoheader.biWidth
    << ", biHeight=" << m_infoheader.biHeight
    << ", biPlanes=" << m_infoheader.biPlanes
    << endl
```

```
    << "  biBitCount="     << m_infoheader.biBitCount
    << ", biCompression=" << m_infoheader.biCompression
    << ", biSizeImage="   << m_infoheader.biSizeImage
    << endl

    << "  biXPelsPerMeter=" << m_infoheader.biXPelsPerMeter
    << ", biYPelsPerMeter=" << m_infoheader.biYPelsPerMeter
    << ", biClrUsed="       << m_infoheader.biClrUsed
    << ", biClrImportant="  << m_infoheader.biClrImportant
    << endl;
}
#endif

/* vim: set et sw=2 ts=2: */
```

4.9 cpixmap.h

```
/**
 * @module CPixmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CFile CBitmap handling XPM.
 * @date   27.04.2009
 */

#ifndef CPixmap_H
#define CPixmap_H

#include <stdint.h>
#include "cbitmap.h"

#define PIXMAP_IDENTIFIER "/* XPM */"
#define PIXMAP_COLORCHARS ".#abcdefghijklmnopqrstuvwxyzABCD" \
                          "EFGHIJKLMNOPQRSTUVWXYZ0123456789"

/**
 * @class CPixmap
 * @brief Implementation of CFile handling Pixmap file format.
 *
 * In order to support operations on pixmaps in color mode an
 * implementations of CPixelFormat is used. These classe are
 * allowed to modify the pixmap header, pixelbuffer and color table directly.
 *
 * On error CFile::FileError is thrown.
 */
class CPixmap : public CBitmap
{
  public:
    /**
     * @method CPixmap
     * @brief  Default ctor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixmap();

    /**
     * @method ~CPixmap
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CPixmap()
      {}

    /**
     * @method read
     * @brief  Reads Pixmap from filestream.
     *         On error an exception is thrown.
     * @param  in filestream to read data from
     * @return -
     * @globalvars none
     * @exception CFile::FileError
```

```
     * @exception bad_alloc
     * @conditions none
     */
    void read(std::ifstream& in);

    /**
     * @method write
     * @brief  Writes Pixmap to filestream.
     * @param  out filestream to read data from
     * @return -
     * @globalvars none
     * @exception  FileError
     * @exception  bad_alloc
     * @conditions none
     */
    void write(std::ofstream& out);

#ifdef DEBUG
    /**
     * @method dump
     * @brief  Dumps the Pixmap file header and pixel data to ostream
     * @param  out output stream
     * @return -
     * @globalvars
     * @exception
     * @conditions
     */
    void dump(std::ostream& out);
#endif

    /**
     * @method getPixelDataSize
     * @brief  Return size of pixelbuffer
     * @param  -
     * @return size of pixelbuffer
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const uint32_t getPixelDataSize()
    {
      return m_fileheader.width * m_fileheader.height * sizeof(uint32_t);
    }

    /**
     * @method getHeight
     * @brief  Return height of bitmap in pixel
     * @param  -
     * @return height of bitmap in pixel
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    const uint32_t getHeight()
    {
      return m_fileheader.height;
    }

    /**
     * @method getWidth
     * @brief  Return width of bitmap in pixel
     * @param  -
     * @return width of bitmap in pixel
     * @globalvars none
```

```
   * @exception  none
   * @conditions none
   */
  const uint32_t getWidth()
  {
    return m_fileheader.width;
  }

  /**
   * @method hasColorTable
   * @brief  Check if bitmap has a colortable
   *         (we don't support this yet for windows bitmaps)
   * @param  -
   * @return true if bitmap has a colortable. false otherwise
   * @globalvars none
   * @exception  none
   * @conditions none
   */
  const bool hasColorTable()
  {
    return true;
  }

  /**
   * @method isMirrored
   * @brief  Windows Bitmaps can be stored upside down
   * @param  -
   * @return true if bitmap is stored upside down. false otherwise
   * @globalvars none
   * @exception  none
   * @conditions none
   */
  const bool isMirrored()
  {
    /* pixmap is never mirrored */
    return false;
  }

protected:
  /**
   * @method getLine
   * @brief  read trimmed line (terminated by \n) from filestream
   * @param  in               filestream to read data from
   * @param  ignore_comments true: ignore c-like comments
   * @return return trimmed line from filestream
   * @globalvars none
   * @exception  none
   * @conditions none
   */
  std::string getLine(std::ifstream& in, bool ignore_comments = true);

  /**
   * @method getArrayLine
   * @brief  read trimmed c-arrayline from filestream
   * @param  in filestream to read data from
   * @return return trimmed c-arrayline from filestream
   * @globalvars none
   * @exception  FileError
   * @conditions none
   */
  std::string getCArrayLine(std::ifstream& in);

  /**
   * @method getXPMColorID
```

```
     * @brief  get xpm color identifier, generated using an index
     * @param  index  index used to generate the xpm color identifier
     * @param  length length of xpm color identifier
     * @return return xpm color identifier, generated using index
     * @globalvars none
     * @exception  FileError
     * @conditions none
     */
    const std::string getXPMColorID(unsigned int index, unsigned int length);

    /**
     * @brief Pixmap Header structure
     */
    typedef struct
    {
      /** the xpm width in pixels (signed integer) */
      uint32_t width;
      /** the xpm height in pixels (signed integer) */
      uint32_t height;
      /** the number of colors (signed integer) */
      uint32_t nColor;
      /** the number of characters per pixel (signed integer) */
      uint32_t nChar;
      /** X-Position Hotspots */
      uint32_t xHotspot;
      /** Y-Position Hotspots */
      uint32_t yHotspot;
      /** is hotspot set */
      bool _HOTSPOT;
      /** XPMEXT extension tag found*/
      bool _XPMEXT;
      /** unchanged extension */
      std::string extension;
    } PIXMAP_FILEHEADER;

    /* members */
    /** fileheader */
    PIXMAP_FILEHEADER m_fileheader;
    /** name of image/c-array */
    std::string m_imagename;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.10 cpixmap.cpp

```
/**
 * @module CPixmap
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CFile handling Windows Bitmaps.
 * @date   27.04.2009
 */

#include <sstream>
#include <iomanip>
#include <vector>
#include <boost/algorithm/string/split.hpp>
#include <boost/algorithm/string.hpp>
#include <boost/lexical_cast.hpp>
#include <assert.h>
#ifdef DEBUG
# include <iostream>
#endif
#include "cpixmap.h"
#include "cpixelformat_indexed8.h"

using namespace std;
using namespace boost;

CPixmap::CPixmap()
  : m_imagename("")
{
  m_types.insert("XPM");

  /* add our handlers */
  m_handlers.insert(new CPixelFormat_Indexed8(this));
}

/*-----------------------------------------------------------------------------*/

std::string CPixmap::getLine(std::ifstream& in, bool ignore_comments)
{
  string line("");
  while(!in.eof() && in.good())
  {
    getline(in, line);
    trim(line);
    /* ignore simple ansi c comments. only one-liners */
    if (ignore_comments && line.find_first_of("/*") == 0)
      continue;
    if (!line.empty())
      break;
  }
  return line;
}

/*-----------------------------------------------------------------------------*/

std::string CPixmap::getCArrayLine(std::ifstream& in)
{
  string line = getLine(in, true);
  if (line.empty())
    return line;

  /* this stuff isn't correct too, but we are no c-parser anyway */
  if (line[0] != '"')
    throw FileError("Pixmap array has invalid c-syntax.");
```

```
  if (line[line.length() - 1] != ',' && line[line.length() - 1] != '}')
    throw FileError("Pixmap array has invalid c-syntax.");
  size_t end = line.find_last_of("\"");
  if (end == string::npos)
    throw FileError("Pixmap array has invalid c-syntax.");

  return line.substr(1, end - 1);
}

/*-------------------------------------------------------------------------*/

void CPixmap::read(std::ifstream& in)
{
  /*
   * <C-Comment> XPM <C-Comment>
   * static char * <pixmap_name>[] = {
   * <Values>
   * <Colors>
   * <Pixels>
   * <Extensions>
   * };
   */

  string line, tmp;
  stringstream istr;
  std::vector<std::string> list;
  m_fileheader._XPMEXT  = false;
  m_fileheader._HOTSPOT = false;

  /* get pixelformat instance first */
  m_pixelformat = NULL;
  set<CPixelFormat *>::iterator it;
  for (it = m_handlers.begin(); it != m_handlers.end(); it++)
  {
    /* we only have one! */
    m_pixelformat = *it;
    break;
  }
  if (m_pixelformat == NULL)
    throw FileError("Pixmap color mode is not supported.");

  /* first line has to be PIXMAP_IDENTIFIER */
  line = getLine(in, false);
  if (line != PIXMAP_IDENTIFIER)
    throw FileError("Pixmap has no identifier.");

  /* second line is a c array. we don't do much syntax checking */
  line = getLine(in);
  istr.str(line);
  while(m_imagename.empty() && !istr.eof() && istr.good())
  {
    size_t end;
    istr >> tmp;
    if ((end = tmp.find_first_of("[]")) != string::npos)
    {
      /* <xxx>*<imagename>[]<yyy> */
      size_t start = tmp.find_first_of('*');
      start = (start == string::npos) ? 0 : start + 1;
      m_imagename = tmp.substr(start, end - start);
    }
  }
  if (m_imagename.empty())
    throw FileError("Pixmap has no imagename.");
```

```
/* additional: check "{" exists */
assert(!line.empty());
if (line[line.length() - 1] != '{')
  throw FileError("Pixmap array has no opening bracket.");

/* parse <Values>-section */
line = getCArrayLine(in);
if (line.empty())
  throw FileError("Pixmap has no Values-section.");
algorithm::split(list, line, is_any_of(" \t"));
if (list.size() < 4)
  throw FileError("Pixmap has invalid Values-section.");
try
{
  m_fileheader.width  = lexical_cast<uint32_t>(list[0]);
  m_fileheader.height = lexical_cast<uint32_t>(list[1]);
  m_fileheader.nColor = lexical_cast<uint32_t>(list[2]);
  m_fileheader.nChar  = lexical_cast<uint32_t>(list[3]);

  if (list.size() > 4)
  {
    if (list.size() >= 6)
    {
      m_fileheader._HOTSPOT = true;
      m_fileheader.xHotspot = lexical_cast<uint32_t>(list[4]);
      m_fileheader.yHotspot = lexical_cast<uint32_t>(list[5]);
    }
    if (list.size() != 6)
    {
      if (list[list.size() - 1] != "XPMEXT")
        throw FileError("Unknown parameter count in Values-Section.");
      else
        m_fileheader._XPMEXT = true;
    }
  }
}
catch(bad_lexical_cast& ex)
{
  throw FileError("Value of Values-section is invalid: " + string(ex.what()));
}

/* parse <Colors>-table */
string character;
/* map[id][colortype] = color */
map<string, map<string, CPixelFormat::RGBPIXEL *> > colors;
/* map[id] = indices */
map<string, uint32_t> colornr;
uint32_t index = 0;
for(uint32_t i = 0; i < m_fileheader.nColor; i++)
{
  line = getCArrayLine(in);
  if (line.empty())
    throw FileError("Pixmap has missing colortable-entry.");
  algorithm::split(list, line, is_any_of(" \t"));
  if (list.size() < 3)
    throw FileError("Pixmap colortable-entry is invalid.");

  /* read pixel character */
  character = list[0];
  if (character.length() != m_fileheader.nChar)
    throw FileError("Pixmap colorcharacter is invalid.");
  if (colors.find(character) != colors.end())
    throw FileError("Duplicate colorcharacter found.");
```

```
  /* read colors */
  if ((list.size() - 1) % 2 != 0)
    throw FileError("Pixmap color entrys are invalid.");
  for(uint32_t j = 1; j < list.size(); j = j + 2)
  {
    /* we only support hex-color notations */
    if (list[j + 1].length() != 7)
      throw FileError("Pixmap color value is invalid.");
    if (list[j + 1].at(0) != '#')
      throw FileError("Pixmap color table value is not hexadecimal.");

    /* we only support c-colors! - remove only if you free the pixels */
    if (list[j] != "c")
      continue;

    CPixelFormat::RGBPIXEL *pixel = new CPixelFormat::RGBPIXEL;
    pixel->red   = strtoul(list[j + 1].substr(1, 2).c_str(), NULL, 16);
    pixel->green = strtoul(list[j + 1].substr(3, 2).c_str(), NULL, 16);
    pixel->blue  = strtoul(list[j + 1].substr(5, 2).c_str(), NULL, 16);
    colors[ character ][ list[j] ] = pixel;
  }

  /* we only support c-colors! */
  if (colors[ character ].find("c") == colors[ character ].end())
    throw FileError("Pixmap color entry has missing c-value.");

  /* add pixel to colortable */
  colornr[ character ] = index;
  m_colortable[ index ] = colors[ character ]["c"];
  index++;
}

/* read pixel data */
if (getPixelDataSize() > 0)
{
  if (m_pixeldata != NULL)
    delete[] m_pixeldata;
  m_pixeldata = new uint8_t[getPixelDataSize()];

  for (uint32_t y = 0; y < getHeight(); y++)
  {
    line = getCArrayLine(in);
    if (line.empty())
      throw FileError("Pixmap has no pixel data.");
    if (line.length() != getWidth())
      throw FileError("Pixmap pixeldata width is larger than header width.");

    /* convert color identifier to our own identifiers */
    for(uint32_t x = 0; x < getWidth(); x++)
    {
      character = line.substr(x * m_fileheader.nChar, m_fileheader.nChar);
      assert(!character.empty());
      if (colornr.find(character) == colornr.end())
        throw FileError("Pixel has no reference in colortable.");

      uint32_t offset = y * getWidth() + x;

      /* boundary check */
      if (offset * sizeof(uint32_t) + sizeof(uint32_t) > getPixelDataSize())
        throw FileError("Pixel position is out of range.");

      *((uint32_t *)m_pixeldata + offset) = colornr[ character ];
    }
  }
```

```
  }

  /* get extension */
  if (m_fileheader._XPMEXT)
    getline(in, m_fileheader.extension, '}');
  if (!in.good())
    throw FileError("Pixmap array isn't closed properly.");

  /* set rowsize */
  m_rowsize = sizeof(uint32_t) * getWidth();
}

/*----------------------------------------------------------------------------*/

const std::string CPixmap::getXPMColorID(unsigned int index, unsigned int length)
{
  static const char code[] = PIXMAP_COLORCHARS;
  assert(strlen(code) > 0);
  assert(length > 0);
  string str("");
  for(unsigned int i = length - 1; i > 0; i--)
  {
    str += code[index % strlen(code)];
    index /= strlen(code);
  }
  str += code[index];
  assert(!str.empty());
  return str;
}

/*----------------------------------------------------------------------------*/

void CPixmap::write(std::ofstream& out)
{
  m_fileheader.nColor = m_colortable.size();
  m_fileheader.nChar  = m_fileheader.nColor / strlen(PIXMAP_COLORCHARS) + 1;

  /* header comment */
  out << PIXMAP_IDENTIFIER << endl;

  /* variables*/
  assert(!m_imagename.empty());
  out << "static char *" << m_imagename << "[] = {" << endl;
  out << "\"" << m_fileheader.width << " " << m_fileheader.height
      << " " << m_fileheader.nColor << " " << m_fileheader.nChar;

  /* optional values */
  if (m_fileheader._HOTSPOT)
    out << " " << m_fileheader.xHotspot << " " << m_fileheader.yHotspot;
  if (m_fileheader._XPMEXT)
    out << " " << "XPMEXT";
  out << "\"," << endl;

  /* color table */
  map<uint32_t, CPixelFormat::RGBPIXEL *>::iterator it;
  for (it = m_colortable.begin(); it != m_colortable.end(); it++)
  {
    out << "\"" << getXPMColorID((*it).first, m_fileheader.nChar);
    /* we only support c-colors! */
    out << "\tc #";
    out << setfill('0') << setw(2) << hex << uppercase << (*it).second->red
        << setfill('0') << setw(2) << hex << uppercase << (*it).second->green
        << setfill('0') << setw(2) << hex << uppercase << (*it).second->blue;
    out << "\"," << endl;
```

```
  }

  /* pixel data */
  for (uint32_t y = 0; y < getHeight(); y++)
  {
    out << "\"";
    for(uint32_t x = 0; x < getWidth(); x++)
    {
      uint32_t offset = y * getWidth() + x;

      /* boundary check */
      if (offset * sizeof(uint32_t) + sizeof(uint32_t) > getPixelDataSize())
        throw FileError("Pixel position is out of range.");

      uint32_t color = *((uint32_t *)m_pixeldata + offset);

      if ((it = m_colortable.find(color)) == m_colortable.end())
        throw FileError("Pixel has no reference in colortable.");
      out << getXPMColorID((*it).first, m_fileheader.nChar);
    }
    out << "\"," << endl;
  }

  /* extension */
  if (m_fileheader._XPMEXT)
    out << m_fileheader.extension;

  out <<"};";
}

/*-----------------------------------------------------------------------------*/

#ifdef DEBUG
void CPixmap::dump(std::ostream& out)
{
  /* values*/
  cout << "[XPM Header Values]" << endl
       << "width="      << m_fileheader.width      << endl
       << "height="     << m_fileheader.height     << endl
       << "nColor="     << m_fileheader.nColor     << endl
       << "nChar="      << m_fileheader.nChar      << endl
       << "Hotspot="    << m_fileheader.xHotspot   << endl
       << "yHotspot="   << m_fileheader.yHotspot   << endl
       << "_HOTSPOT="   << m_fileheader._HOTSPOT   << endl
       << "_XPMEXT="    << m_fileheader._XPMEXT    << endl
       << "extension="  << m_fileheader.extension  << endl
       << endl;

  /* colors*/
  map<uint32_t, CPixelFormat::RGBPIXEL *>::iterator it;
  cout << "[Color Table]" << endl;
  for (it = m_colortable.begin(); it != m_colortable.end(); it++)
  {
    out << (*it).first << ": "
        << setfill('0') << setw(3) << (*it).second->red   << " "
        << setfill('0') << setw(3) << (*it).second->green << " "
        << setfill('0') << setw(3) << (*it).second->blue  << " "
        << endl;
  }
}
#endif

/* vim: set et sw=2 ts=2: */
```

4.11 cpixelformat.h

```
/**
 * @module cpixelformat
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Abstract class for handling different color bitcount of Bitmaps.
 *         Needed for generic use in CBitmap.
 * @date   18.04.2009
 */

#ifndef CPIXELFORMAT_H
#define CPIXELFORMAT_H

#include <fstream>
#include <stdexcept>

class CBitmap;

/**
 * @class CPixelFormat
 * @brief Abstract class for handling different color bitcount of Bitmaps.
 *
 * Needed for generic use in CBitmap.
 *
 * On error throw PixelFormatError.
 */
class CPixelFormat
{
  public:
    /**
     * @class PixelFormatError
     * @brief Exception thrown by implemententations of CPixelFormat
     */
    class PixelFormatError : public std::invalid_argument {
      public:
        /**
         * @method PixelFormatError
         * @brief  Default exception ctor
         * @param  what message to pass along
         * @return -
         * @globalvars none
         * @exception  none
         * @conditions none
         */
        PixelFormatError(const std::string& what)
          : std::invalid_argument(what)
        {}
    };

    /**
     * @method CPixelFormat
     * @brief  Default ctor
     * @param  bitmap pointer to CBitmap instance
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixelFormat(CBitmap *bitmap)
      : m_bitmap(bitmap)
    {}

    /**
```

```
 * @method ~CPixelFormat
 * @brief  Default dtor (virtual)
 * @param  -
 * @return -
 * @globalvars none
 * @exception  none
 * @conditions none
 */
virtual ~CPixelFormat()
{};

/**
 * @brief RGB Pixel structure
 */
typedef struct
{
  /** red */
  uint32_t red;
  /** green */
  uint32_t green;
  /** blue */
  uint32_t blue;
} RGBPIXEL;

/**
 * @method getPixel
 * @brief  Get pixel at coordinates x, y
 * @param  pixel reference to pixel data
 * @param  x     x-coordinate
 * @param  y     y-coordinate
 * @return -
 * @globalvars none
 * @exception  PixelFormatError
 * @conditions none
 */
virtual void getPixel(RGBPIXEL& pixel, const uint32_t x, const uint32_t y) = 0;

/**
 * @method setPixel
 * @brief  Modifies pixel at coordinates x, y
 * @param  pixel reference to new pixel data
 * @param  x     x-coordinate
 * @param  y     y-coordinate
 * @return -
 * @globalvars none
 * @exception  PixelFormatError
 * @conditions none
 */
virtual void setPixel(const RGBPIXEL& pixel, const uint32_t x, const uint32_t y
    ) = 0;

/**
 * @method getBitCount
 * @brief  returns color bitcount supported by this class
 * @param  -
 * @return color bitcount supported by this class
 * @globalvars none
 * @exception  none
 * @conditions none
 */
virtual uint32_t getBitCount() = 0;

/**
 * @method getMaxColor
```

```
     * @brief  Get maximum values for RGB pixel
     * @param  pixel reference to pixel struct
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    virtual void getMaxColor(RGBPIXEL& pixel) = 0;

  protected:
    /* members */
    /** pointer to CBitmap instance */
    CBitmap *m_bitmap;
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.12 cpixelformat_bgr24.h

```
/**
 * @module cpixelformat_bgr24
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 * @date   18.04.2009
 */

#ifndef CPIXELFORMAT_BGR24_H
#define CPIXELFORMAT_BGR24_H

#include <fstream>
#include "cpixelformat.h"

/**
 * @class CPixelFormat_BGR24
 * @brief Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 *
 * On error CPixelFormat::PixelFormatError is thrown.
 */
class CPixelFormat_BGR24 : public CPixelFormat
{
  public:
    /**
     * @method CPixelFormat_BGR24
     * @brief  Default ctor
     * @param  bitmap pointer to CBitmap instance
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixelFormat_BGR24(CBitmap *bitmap)
      : CPixelFormat(bitmap)
    {}

    /**
     * @method ~CPixelFormat_BGR24
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CPixelFormat_BGR24()
    {}

    /**
     * @method getPixel
     * @brief  Get pixel at coordinates x, y
     * @param  pixel reference to pixel data
     * @param  x     x-coordinate
     * @param  y     y-coordinate
     * @return -
     * @globalvars none
     * @exception  PixelFormatError
     * @conditions none
     */
    void getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
```

```
     * @method setPixel
     * @brief  Modifies pixel at coordinates x, y
     * @param  pixel reference to new pixel data
     * @param  x     x-coordinate
     * @param  y     y-coordinate
     * @return -
     * @globalvars none
     * @exception  PixelFormatError
     * @conditions none
     */
    void setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
     * @method getBitCount
     * @brief  returns color bitcount supported by this class
     * @param  -
     * @return color bitcount supported by this class
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    uint32_t getBitCount()
    {
      return 24;
    }

    /**
     * @method getMaxColor
     * @brief  Get maximum values for RGB pixel
     * @param  pixel reference to pixel struct
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    void getMaxColor(RGBPIXEL& pixel)
    {
      /* value = 2^8 - 1 */
      pixel.red = pixel.green = pixel.blue = 255;
    }
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.13 cpixelformat_bgr24.cpp

```
/**
 * @module cpixelformat_BGR24
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 * @date   18.04.2009
 */

#include <boost/numeric/conversion/cast.hpp>
#include <assert.h>
#include "cpixelformat_bgr24.h"
#include "cbitmap.h"

using namespace std;

void CPixelFormat_BGR24::getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  assert(m_bitmap->getPixelDataSize() > 0);
  assert(m_bitmap->getRowSize() > 0);

  /* if the y-coordinates are mirrored */
  if (m_bitmap->isMirrored())
    y = m_bitmap->getHeight() - y - 1;
  uint32_t offset = y * m_bitmap->getRowSize() + x * (4 * getBitCount() / 32);

  /* boundary check */
  if (offset + getBitCount()/8 > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  /* get pixel */
  pixel.red   = *(m_bitmap->getPixelData() + offset + 2);
  pixel.green = *(m_bitmap->getPixelData() + offset + 1);
  pixel.blue  = *(m_bitmap->getPixelData() + offset);
}

/*-------------------------------------------------------------------------*/

void CPixelFormat_BGR24::setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  assert(m_bitmap->getPixelDataSize() > 0);
  assert(m_bitmap->getRowSize() > 0);

  /* if the y-coordinates are mirrored */
  if (m_bitmap->isMirrored())
    y = m_bitmap->getHeight() - y - 1;
  uint32_t offset = y * m_bitmap->getRowSize() + x * (4 * getBitCount() / 32);

  /* boundary check */
  if (offset + getBitCount()/8 > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  /* convert color values to correct types */
  uint8_t data[3];
  try
  {
    data[0] = boost::numeric_cast<uint8_t>(pixel.blue);
    data[1] = boost::numeric_cast<uint8_t>(pixel.green);
    data[2] = boost::numeric_cast<uint8_t>(pixel.red);
```

```
  }
  catch(boost::numeric::bad_numeric_cast& ex)
  {
    throw PixelFormatError("Unable to convert pixelcolor to correct size: " +
        string(ex.what()));
  }

  copy(data, data + 3, m_bitmap->getPixelData() + offset);
}

/* vim: set et sw=2 ts=2: */
```

4.14 cpixelformat_bgr555.h

```
/**
 * @module cpixelformat_bgr555
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling BGR555 color (real color)
 *         Windows Bitmaps.
 * @date   26.04.2009
 */

#ifndef CPIXELFORMAT_BGR555_H
#define CPIXELFORMAT_BGR555_H

#include <fstream>
#include "cpixelformat.h"

/* to (un-)pack pixel values */
#define BGR555_RED_SHIFT    10
#define BGR555_GREEN_SHIFT  5
#define BGR555_BLUE_SHIFT   0
#define BGR555_RED_MASK     0x7C00
#define BGR555_GREEN_MASK   0x03E0
#define BGR555_BLUE_MASK    0x001F

/**
 * @class CPixelFormat_BGR555
 * @brief Implementation of CPixelFormat handling BGR555 color (real color)
 *        Windows Bitmaps.
 *
 * On error CPixelFormat::PixelFormatError is thrown.
 */
class CPixelFormat_BGR555 : public CPixelFormat
{
  public:
    /**
     * @method CPixelFormat_BGR55
     * @brief  Default ctor
     * @param  bitmap pointer to CBitmap instance
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixelFormat_BGR555(CBitmap *bitmap)
      : CPixelFormat(bitmap)
    {}

    /**
     * @method ~CPixelFormat_BGR555
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CPixelFormat_BGR555()
    {}

    /**
     * @method getPixel
     * @brief  Get pixel at coordinates x, y
     * @param  pixel reference to pixel data
```

```
     * @param   x       x-coordinate
     * @param   y       y-coordinate
     * @return -
     * @globalvars none
     * @exception   PixelFormatError
     * @conditions none
     */
    void getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
     * @method setPixel
     * @brief  Modifies pixel at coordinates x, y
     * @param  pixel reference to new pixel data
     * @param  x       x-coordinate
     * @param  y       y-coordinate
     * @return -
     * @globalvars none
     * @exception   PixelFormatError
     * @conditions none
     */
    void setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
     * @method getBitCount
     * @brief  returns color bitcount supported by this class
     * @param  -
     * @return color bitcount supported by this class
     * @globalvars none
     * @exception   none
     * @conditions none
     */
    uint32_t getBitCount()
    {
      return 16;
    }

    /**
     * @method getMaxColor
     * @brief  Get maximum values for RGB pixel
     * @param  pixel reference to pixel struct
     * @return -
     * @globalvars none
     * @exception   none
     * @conditions none
     */
    void getMaxColor(RGBPIXEL& pixel)
    {
      /* value = 2^5 - 1 */
      pixel.red = pixel.green = pixel.blue = 31;
    }
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.15 cpixelformat_bgr555.cpp

```
/**
 * @module cpixelformat_BGR555
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling BGR555 color (real color)
 *         Windows Bitmaps.
 * @date   18.04.2009
 */

#include <boost/numeric/conversion/cast.hpp>
#include <assert.h>
#include "cpixelformat_bgr555.h"
#include "cbitmap.h"

using namespace std;

void CPixelFormat_BGR555::getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  assert(m_bitmap->getPixelDataSize() > 0);
  assert(m_bitmap->getRowSize() > 0);

  /* if the y-coordinates are mirrored */
  if (m_bitmap->isMirrored())
    y = m_bitmap->getHeight() - y - 1;
  uint32_t offset = y * m_bitmap->getRowSize() + x * (4 * getBitCount() / 32);

  /* boundary check */
  if (offset + getBitCount()/8 > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  /* get pixel */
  uint8_t *o = m_bitmap->getPixelData() + offset;
  pixel.blue  = (*(uint16_t *)o & BGR555_BLUE_MASK)  >> BGR555_BLUE_SHIFT;
  pixel.green = (*(uint16_t *)o & BGR555_GREEN_MASK) >> BGR555_GREEN_SHIFT;
  pixel.red   = (*(uint16_t *)o & BGR555_RED_MASK)   >> BGR555_RED_SHIFT;
}

/*-------------------------------------------------------------------------*/

void CPixelFormat_BGR555::setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  assert(m_bitmap->getPixelDataSize() > 0);
  assert(m_bitmap->getRowSize() > 0);

  /* if the y-coordinates are mirrored */
  if (m_bitmap->isMirrored())
    y = m_bitmap->getHeight() - y - 1;
  uint32_t offset = y * m_bitmap->getRowSize() + x * (4 * getBitCount() / 32);

  /* boundary check */
  if (offset + getBitCount()/8 > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  /* convert color values to correct types */
  uint8_t *o = m_bitmap->getPixelData() + offset;
  *(uint16_t *)o = (uint16_t)(((pixel.blue << BGR555_BLUE_SHIFT)   &
      BGR555_BLUE_MASK)  |
```

```
                                ((pixel.green << BGR555_GREEN_SHIFT) &
                                    BGR555_GREEN_MASK) |
                                ((pixel.red << BGR555_RED_SHIFT)     &
                                    BGR555_RED_MASK));
}

/* vim: set et sw=2 ts=2: */
```

4.16 cpixelformat_indexed8.h

```
/**
 * @module cpixelformat_bgr24
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling 24bit indexed bitmaps.
 * @date   02.05.2009
 */

#ifndef CPixelFormat_Indexed8_H
#define CPixelFormat_Indexed8_H

#include <fstream>
#include "cpixelformat.h"

/**
 * @class CPixelFormat_Indexed8
 * @brief Implementation of CPixelFormat handling 24bit indexed bitmaps.
 *
 * On error CPixelFormat::PixelFormatError is thrown.
 */
class CPixelFormat_Indexed8 : public CPixelFormat
{
  public:
    /**
     * @method CPixelFormat_Indexed8
     * @brief  Default ctor
     * @param  bitmap pointer to CBitmap instance
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixelFormat_Indexed8(CBitmap *bitmap)
      : CPixelFormat(bitmap)
    {}

    /**
     * @method ~CPixelFormat_Indexed8
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CPixelFormat_Indexed8()
    {}

    /**
     * @method getPixel
     * @brief  Get pixel at coordinates x, y
     * @param  pixel reference to pixel data
     * @param  x     x-coordinate
     * @param  y     y-coordinate
     * @return -
     * @globalvars none
     * @exception  PixelFormatError
     * @conditions none
     */
    void getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
```

```
     * @method setPixel
     * @brief  Modifies pixel at coordinates x, y
     * @param  pixel reference to new pixel data
     * @param  x     x-coordinate
     * @param  y     y-coordinate
     * @return -
     * @globalvars none
     * @exception  PixelFormatError
     * @conditions none
     */
    void setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y);

    /**
     * @method getBitCount
     * @brief  returns the bitcount needed for indexing the color tabel.
     * @param  -
     * @return bitcount of indexes supported by this class
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    uint32_t getBitCount()
    {
      return 24;
    }

    /**
     * @method getMaxColor
     * @brief  Get maximum values for RGB pixel
     * @param  pixel reference to pixel struct
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    void getMaxColor(RGBPIXEL& pixel)
    {
      /* value = 2^8 - 1 */
      pixel.red = pixel.green = pixel.blue = 255;
    }
};

#endif

/* vim: set et sw=2 ts=2: */
```

4.17 cpixelformat_indexed8.cpp

```
/**
 * @module CPixelFormat_Indexed8
 * @author Guenther Neuwirth (0626638), Manuel Mausz (0728348)
 * @brief  Implementation of CPixelFormat handling 24bit indexed bitmaps.
 * @date   02.05.2009
 */

#include <boost/numeric/conversion/cast.hpp>
#include <assert.h>
#include "cpixelformat_indexed8.h"
#include "cbitmap.h"

using namespace std;

void CPixelFormat_Indexed8::getPixel(RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  if (m_bitmap->getColorTable().size() == 0)
    return;
  assert(m_bitmap->getPixelDataSize() > 0);

  uint32_t offset = y * m_bitmap->getWidth() + x;

  /* boundary check */
  if (offset * sizeof(uint32_t) + sizeof(uint32_t) > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  uint32_t color = *((uint32_t *)m_bitmap->getPixelData() + offset);

  map<uint32_t, RGBPIXEL *>::iterator it;
  if ((it = m_bitmap->getColorTable().find(color)) == m_bitmap->getColorTable().end
      ())
    throw PixelFormatError("Pixel has no reference in colortable.");

  pixel.red   = (*it).second->red;
  pixel.green = (*it).second->green;
  pixel.blue  = (*it).second->blue;
}

/*-------------------------------------------------------------------------------*/

void CPixelFormat_Indexed8::setPixel(const RGBPIXEL& pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");
  /* if colortable is empty there are no pixels */
  if (m_bitmap->getColorTable().size() == 0)
    return;
  assert(m_bitmap->getPixelDataSize() > 0);

  uint32_t offset = y * m_bitmap->getWidth() + x;

  /* boundary check */
  if (offset * sizeof(uint32_t) + sizeof(uint32_t) > m_bitmap->getPixelDataSize())
    throw PixelFormatError("Pixel position is out of range.");

  /* try to look up color in colortable */
  map<uint32_t, RGBPIXEL *>::iterator it;
  for(it = m_bitmap->getColorTable().begin(); it != m_bitmap->getColorTable().end()
      ; it++)
```

```
  {
    if ((*it).second->red   == pixel.red &&
        (*it).second->green == pixel.green &&
        (*it).second->blue  == pixel.blue)
      break;
  }

  uint32_t index = (*it).first;
  /* need to get a new entry for our color */
  if (it == m_bitmap->getColorTable().end())
  {
    index = (*it).first + 1;
    RGBPIXEL *pixelptr = new RGBPIXEL;
    pixelptr->red   = pixel.red;
    pixelptr->green = pixel.green;
    pixelptr->blue  = pixel.blue;
    m_bitmap->getColorTable()[ index ] = pixelptr;
  }

  /* set color */
  *((uint32_t *)m_bitmap->getPixelData() + offset) = index;
}

/* vim: set et sw=2 ts=2: */
```