Object-Oriented Programming VL

Laborprotokoll

Beispiel 1

Manuel Mausz, Matr. Nr.0728348
manuel-tu@mausz.at

Wien, am 20. April 2009

Inhaltsverzeichnis

1 Aufgabenstellung - Beispiel 1

Beispielangaben zu
OOP (Objekt-Orientierte Programmierung)
SS 2009

Beispiel 1

Relevante Themen:

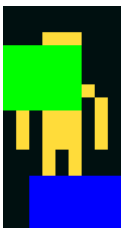
- Ressourcen-Allokation
- Fehlerbehandlung
- Dynamic Objects
- Klassen zur Informationskapselung
- Standard I/O von C++

Design und Implementierung:
Schreiben Sie in C++ ein skriptgesteuertes Bildbearbeitungsprogramm imgsynth. Das Programm liest als einzige Kommandozeilenoption den Namen einer Skriptdatei ein, welche die Befehle, die vom Programm durchzuführen sind, beinhaltet. Jede Zeile in dieser Skriptdatei enthält genau einen kompletten Befehl. Diese Befehle werden sequentiell abgearbeitet, wobei jeder Befehl auf den Bildspeicher angewendet wird. Der erste Befehl muss allerdings immer ein Lesebefehl sein, um eine Bilddatei vom Dateisystem einzulesen und der letzte Befehl muss immer ein Schreibbefehl sein, um die aktuellen Bilddaten in eine Datei zu schreiben. Jeder Befehl kann eine fix definierte Anzahl von Argumenten übernehmen.

Die Synopsis des Programms ist folgende: imgsynth -i <scriptfile>

Beispiel: Angenommen, imgsynth wird aufgerufen mit imgsynth -i bsp.isc, wobei die Skriptdatei bsp.isc folgende Befehle enthält:

```
read(BMP, yellow_man.bmp)
fillrect(0,3,6,5,0,255,0)
fillrect(2,13,7,4,0,0,255)
write(BMP, "yellow_man f.bmp")
```



... so wird damit die Bilddatei yellow_man.bmp (Bild links) eingelesen, weiters werden zwei Rechtecke in das Bild gemalt und schlussendlich wird das modifizierte Bild unter dem Filenamen "yellow_man f.bmp" (Bild rechts) gespeichert. White spaces innerhalb von Dateinamen erfordern, dass der Dateiname mit Anführungszeichen umgeben wird.

Im folgendem werden die zu unterstützenden Befehle nochmals genauer erklärt:

read(typ, name) … Öffnet eine Grafikdatei „name“ und interpretiert das Format entsprechend dem spezifizierten Typ „typ” und liest den Bildinhalt in den Hauptspeicher. Der Name der Grafikdatei kann optional unter Anführungszeichen gestellt sein. Als Typ muss nur der Dateityp „BMP” unterstützt werden, was für eine vereinfachte Version des Windows Bitmap Dateiformates (auch als OS/2 Bitmap Dateiformat bekannt) steht (siehe Beschreibung am Beispielende).

fillrect(x, y, w, h, r, g, b) … Zeichnet ein gefülltes Rechteck an die Koordinate (x,y), mit der angegebenen Größe (w, h) und der angegebenen Farbe (r, g, b). Die Koordinate (0,0) bezeichnet dabei die Bildecke „links oben”. Die Größe (w, h) beschreibt die Breite „w“ (nach rechts) und die Höhe „h“ (nach unten) des zu malenden Rechteckes in Pixel. Die Farbe (r, g, b) gibt die Farbanteile Rot, Grün und Blau jeweils im Wertebereich von 0..255 der Füllfarbe des Rechteckes an.

write(typ, name) … Speichert das aktuelle Bild als Datei „name“ mit Typ „typ“ im Dateisystem. Als Formatstring braucht nur „BMP“ unterstützt werden, was dieselbe Bedeutung hat wie beim Befehl read().

Whitespaces wie beispielsweise Leerzeichen und Tabulatoren sind beim Einlesen der Skriptdatei zu ignorieren.

Erstellen Sie dazu eine Klasse CBitmap, welche von einem Eingabestream Grafiken vom Format „BMP“ dekodieren kann. Der benötigte Bildspeicher ist dabei dynamisch anzufordern. Falls der Eingabestream nicht dem Bildformat entspricht, soll eine Fehlerbehandlung basierend auf Ausnahmen (Exceptions) eingebaut werden. Da es im Allgemeinen unterschiedliche Pixelformate gibt (Anzahl der Bits pro Farbkanal), soll das Abspeichern der Bildpunkte in den Bildspeicher über eine separate Klasse CPixelFormat durchgeführt werden. Der pro Pixel allokierte Speicher soll dem konkreten Pixelformat entsprechen und nicht beispielsweise allgemein als eine 32-bit-Zahl gespeichert werden. Die Klasse CPixelFormat soll beispielsweise eine Methode setPixel(char* data, x, y) enthalten, wobei „data“ ein Zeiger auf den Farbwert darstellt und „x“ und „y“ die Pixelkoordination darstellen, wo das Pixel abgespeichert werden soll. Die Klasse CPixelFormat ist in die Klasse CBitmap mittels Komposition einzubinden.

Der read-Befehl soll die Pixel nicht auf ein gemeinsames Pixelformat konvertieren. Die einzelne Filter/Befehle sollen die Operationen direkt im ursprünglichen Pixelformat durchführen.

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

2 Beispiel 1

2.1 Design

Abbildung 1 zeigt das Klassendiagramm der Aufgabe.

Die Klasse CScriptParser übernimmt das Parsen der per Commandlineparameter übergebenen Scriptdatei als String, wobei dieser bereits im Konstruktor übergeben werden muss. Zum Anstoßen des Parsen dient die Funktion parse(). Der (simple) Parser arbeitet Zeilenbasiert und erwartet pro Zeile einen Funktionsaufruf im Syntax: funktionsname(param1, ... paramX). Schachteln von Funktionen ist nicht erlaubt. Der erste Befehl eines Blocks muss der Befehl “read” sein, der Befehl “write” beendet einen Block. Alle Funktionsparameter werden zusammen in einer Liste gespeichert und den entsprechenden Funktionen übergeben. Whitespaces und Anführungszeichen werden bereits vorab gelöscht.

Tritt ein Fehler während des Parsens auf, werden Instanzen der Klasse CSriptError::ParserError als Exception geworfen. Über die Methode getLine() der Exception kann die aktuelle Zeile der Scriptdatei, die den Fehler erzeugt hat, ausgelesen werden.

Da der Scriptbefehl “read” und “write” einen Dateitypparameter enthält, ist es potentiell möglich verschiedene Dateitypen zu öffnen und zu bearbeiten. Um dies aus der Sicht des Parsers generisch durchzuführen, müssen alle Klassen, die Dateioperationen durchführen können, von der Abstrakten Klasse CFile abgeleitet sein und somit mindestens dessen virtuelle Methoden implementieren. Alle Implementation müssen weiters im Konstruktor die unterstützten Dateitypen in die Membervariable m_types hinzufügen, damit der Parser die jeweils zuständige Implementation verwenden kann.

Die Methode callFunc(...) dient zum Aufruf von dateitypspezifische Scriptfunktionen. Hierzu übergibt der Parser automatisch alle unbekannten Funktionen und dessen Parameter innerhalb eines Block (abgesehen von “read” und “write”) der jeweils zuständigen Instanz.

Zur Fehlerbehandlung sollen Implementationen von CFile Instanzen der Klasse CFile::FileError als Exception werfen. Diese werden vom Parser gefangen und in Instanzen der eigenen Exception des Parsers CSriptError::ParserError übersetzt.

Die Klasse CBitmap implementiert die Abstrakte Klasse CFile und kann Dateien des Types “BMP” (Windows Bitmap) bearbeiten. Beim Lesen der Datei werden rudimentäre Checks des Dateiheaders durchgeführt. Der Speicher der Pixel, wird wie gewünscht dynamisch alloziert. Um aber die verschiedenen möglichen Farbtiefen von Windows Bitmap zu unterstützen, werden die Schreiboperationen

auf die Pixeldaten an eine Instanz der je nach Farbtiefe zuständigen Implementation der Abstrakten Klasse CPixelFormat delegiert. Diese Instanz wird während der Analyse des Dateiheaders ebenfalls dynamisch allokiert.

Damit Implementationen der abstrakten Klasse CPixelFormat direkt auf die Daten des Windows Bitmap zugreifen können, wird im Konstruktor ein Pointer auf die Instanz von CBitmap übergeben. Über die Public Getter-Methoden von CBitmap erfolgt der direkt Zugriff. Fehlerbehandlung erfolgt Exceptions der Klasse CPixelFormat::PixelFormatError.

2.2 Verwaltung der Ressourcen

Alle Klassen, die im Laufe ihrer Existenz Ressourcen dynamische allozieren, initialisieren die jeweiligen Membervariablen im Konstruktor auf NULL und geben diese, sofern tatsächlich alloziert, im spätestens Destruktor wieder frei.
Alle Dateien, die geöffnet werden, werden nach dem Abfangen der Exception auch wieder geschlossen, sofern alle möglichen, auftretenden Exceptions (std::bad_alloc ausgenommen) auch vorab übersetzt wurden.

2.3 Fehlerbehandlung

Alle Implementationen der abstrakten Klasse CPixelFormat werfen Exceptions der Klasse CPixelFormat::PixelFormatError. Diese werden von CBitmap gefangen und in Exceptions der Klasse CFile::FileError übersetzt, welche wiederum von der Klasse CScriptParser gefangen und in Exceptions der Klasse CScriptParser::ParserError übersetzt werden.
Diese Exceptions sowie Exceptions des Typs std::exception werden schlussendlich vom Hauptprogramm gefangen und geben eine entsprechende Fehlermeldung an den Benutzer auf stderr aus.

2.4 Implementierung

Siehe Punkt 2.1 und Abbildung 1 sowie Punkt 4.

Alle Exceptions wurden von std::invalid_argument abgeleitet und der Konstruktor gemäß den üblichen Konventionen implementiert:

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

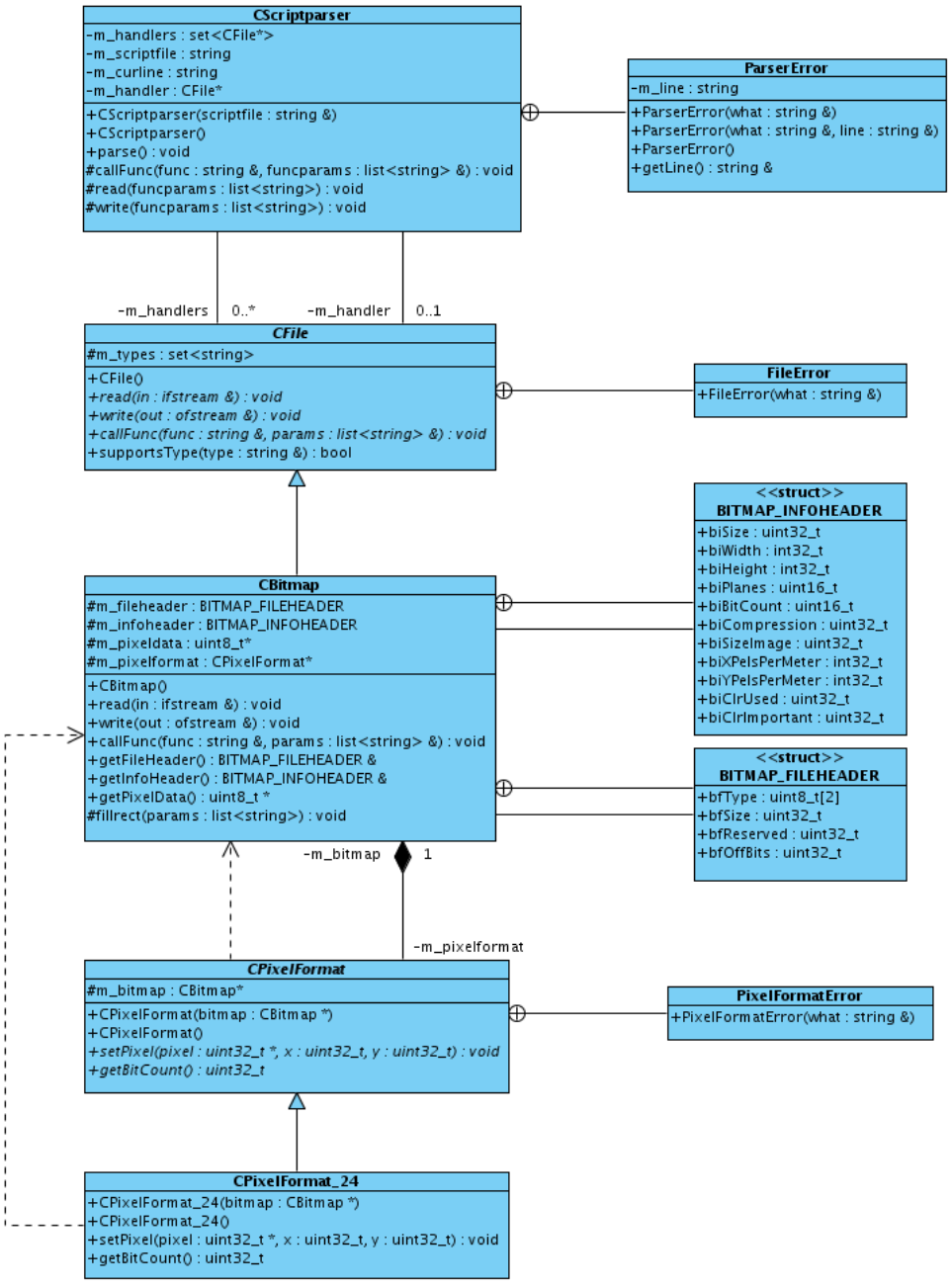


Abbildung 1: Klassendiagramm 1

3 Projektverlauf

3.1 Probleme und Fallstricke

In der Hoffnung das in der nächsten Aufgabe weitere Funktionen, Dateitypen und/oder Farbtiefen des Windows Bitmaps-Formats verlangt werden, wurden diese sehr

generisch implementiert.
Ursprünglich wollte ich die jeweilig unterstützen Scriptfunktionen mittels std::map<std::string, (void *)()> an den Scriptparser zurückgeben, sodass dieser direkt die jeweilige Methode (per Pointer) aufrufen kann. Dies funktioniert jedoch logischerweise nur bei statischen Methoden. Daher die einfacher Methode über die callFunc-Methoden, die die Parameter an die jeweiligen internen Methoden weiterdelegieren.

Da sich CBitmap und CPixelFormat gegenseitig referenzieren, müssen die jeweiligen Klassen im Headerfile der anderen Klasse vorab deklariert werden. Andernfalls kann der Compiler die Klasse aufgrund der rekursiven Inklusion nicht finden.

3.2 Arbeitsaufwand

Entwicklungsschritt / Meilenstein	Arbeitsaufwand in Stunden
Erstes Design	15 Minuten
Implementierung (und leichte Anpassung des Designs)	1 Tag
Dokumentation (Doxygen) und Überprüfung alle Anforderungen gemäß der Programmierrichtlinien	2 Tage
Erstellung des Protokolls	1 Tag

4 Listings

4.1 imgsynth.cpp

```
/**
 * @module imgsynth
 * @author Manuel Mausz, 0728348
 * @brief  imgsynth 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
 *      1
 */

#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 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 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 "cbitmap.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 CBitmap);
}

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

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

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

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)
      continue;
```

```
    /* 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 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 Manuel Mausz, 0728348
 * @brief  Implementation of CFile handling Windows Bitmaps.
 * @date   17.04.2009
 */

#ifndef CBITMAP_H
#define CBITMAP_H

#include "cfile.h"

class CPixelFormat;
#include "cpixelformat.h"

/**
 * @class CBitmap
 * @brief Implementation of CFile handling Windows 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_types.insert("BMP");
    }

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

    /**
     * @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);

#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

    /**
     * @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)

    /**
     * @method getFileHeader
     * @brief  Returns reference to fileheader structure of bitmap
     * @param  -
     * @return reference to fileheader structure
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    BITMAP_FILEHEADER &getFileHeader()
    {
      return m_fileheader;
    }

    /**
     * @method getInfoHeader
     * @brief  Returns reference to infoheader structure of bitmap
     * @param  -
     * @return reference to infoheader structure
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    BITMAP_INFOHEADER &getInfoHeader()
    {
      return m_infoheader;
    }

    /**
     * @method getPixelData
     * @brief  Returns pointer to pixelbuffer
     * @param  -
     * @return pointer to pixelbuffer
     * @globalvars none
     * @exception  none
```

```
     * @conditions none
     */
    uint8_t *getPixelData()
    {
      return m_pixeldata;
    }

  protected:
    /**
     * @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);

    /* members */
    /** fileheader */
    BITMAP_FILEHEADER m_fileheader;
    /** infoheader */
    BITMAP_INFOHEADER m_infoheader;
    /** pointer to pixelbuffer */
    uint8_t *m_pixeldata;
    /** pointer to CPixelFormat implementation */
    CPixelFormat *m_pixelformat;
};

#endif

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

4.6 cbitmap.cpp

```
/**
 * @module cbitmap
 * @author Manuel Mausz, 0728348
 * @brief  Implementation of CFile handling Windows Bitmaps.
 * @date   17.04.2009
 */

#include <boost/lexical_cast.hpp>
#include <boost/numeric/conversion/cast.hpp>
#ifdef DEBUG
# include <iostream>
#endif
#include "cbitmap.h"
#include "cpixelformat_24.h"

using namespace std;

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

  if (m_pixelformat != NULL)
    delete m_pixelformat;
  m_pixelformat = NULL;
}

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

void CBitmap::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.");

  /* currently only 24bit */
  if (m_infoheader.biBitCount != 24)
    throw FileError("Bitmap bitcount is 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];
    in.read(reinterpret_cast<char *>(m_pixeldata), m_infoheader.biSizeImage);
  }

  /* create pixelformat instance */
  if (m_pixelformat != NULL)
    delete m_pixelformat;
  m_pixelformat = NULL;
  if (m_infoheader.biBitCount == 24)
    m_pixelformat = new CPixelFormat_24(this);
}

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

void CBitmap::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);
}

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

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
    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).");

  /* 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() + ").");
  }

  /* width and height can be negativ */
  uint32_t width  = static_cast<uint32_t>(abs(m_infoheader.biWidth));
  uint32_t height = static_cast<uint32_t>(abs(m_infoheader.biHeight));

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

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

  if (pparams[4] < 0 || pparams[4] > 255
   || pparams[5] < 0 || pparams[5] > 255
   || pparams[6] < 0 || pparams[6] > 255)
    throw FileError("At least one pixel color parameter is out of range.");

  /* call setPixel for every pixel in the rectangel */
  if (m_pixeldata != NULL && m_pixelformat != NULL)
  {
    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(&pparams[4], 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());
        }
      }
    }
  }
}

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

#ifdef DEBUG
void CBitmap::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: */
```

4.7 cpixelformat.h

```
/**
 * @module cpixelformat
 * @author 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;
#include "cbitmap.h"

/**
 * @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 CBitmap
     * @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()
    {};

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

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

#endif

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

4.8 cpixelformat_24.h

```
/**
 * @module cpixelformat_24
 * @author Manuel Mausz, 0728348
 * @brief  Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 * @date   18.04.2009
 */

#ifndef CPIXELFORMAT_24_H
#define CPIXELFORMAT_24_H

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

/**
 * @class CPixelFormat_24
 * @brief Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 *
 * On error CPixelFormat::PixelFormatError is thrown.
 */
class CPixelFormat_24 : public CPixelFormat
{
  public:
    /**
     * @method CPixelFormat_24
     * @brief  Default ctor
     * @param  bitmap pointer to CBitmap instance
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    CPixelFormat_24(CBitmap *bitmap)
      : CPixelFormat(bitmap)
    {}

    /**
     * @method ~CPixelFormat_24
     * @brief  Default dtor
     * @param  -
     * @return -
     * @globalvars none
     * @exception  none
     * @conditions none
     */
    ~CPixelFormat_24()
    {}

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

#endif

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

4.9 cpixelformat_24.cpp

```
/**
 * @module cpixelformat_24
 * @author Manuel Mausz, 0728348
 * @brief  Implementation of CPixelFormat handling 24bit color Windows Bitmaps.
 * @date   18.04.2009
 */

#include <boost/numeric/conversion/cast.hpp>
#include "cpixelformat_24.h"

using namespace std;

void CPixelFormat_24::setPixel(const uint32_t *pixel, uint32_t x, uint32_t y)
{
  if (m_bitmap->getPixelData() == NULL)
    throw PixelFormatError("No pixelbuffer allocated.");

  uint32_t rowsize = 4 * static_cast<uint32_t>(
    ((CPixelFormat_24::getBitCount() * abs(m_bitmap->getInfoHeader().biWidth)) +
        31) / 32
  );

  /* if height is positive the y-coordinates are mirrored */
  if (m_bitmap->getInfoHeader().biHeight > 0)
    y = m_bitmap->getInfoHeader().biHeight - y - 1;
  uint32_t offset = y * rowsize + x * (4 * getBitCount() / 32);

  /* boundary check */
  if (offset + 3 * sizeof(uint8_t) > m_bitmap->getInfoHeader().biSizeImage)
    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[2]);
    data[1] = boost::numeric_cast<uint8_t>(pixel[1]);
    data[2] = boost::numeric_cast<uint8_t>(pixel[0]);
  }
  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: */
```