Directory Mapper
Version 1.0.0

Matthew B. Gately

Table of Contents

Introduction

Directory Mapper is designed to provide an easy way to generate a report detailing all folders and files in a given directory. The only report format available in this version is HTML.

Program Operation

System Requirements

Directory Mapper does not need to be installed in order to run. It comes with the necessary .dll files and only requires Sun Java be installed. As a result it can be run from a flash drive.

Running the Program

To launch the program double click the executable file. Once the program loads you are presented with the window displayed in **Error! Reference source not found.**. To generate the report, follow these steps:

1. Enter the path of the directory to be mapped in the “Root Directory” textbox
2. Enter the path for the HTML report including the report name .html in the “Output File” textbox (see the example below the textbox in Figure 1)
3. Choose whether only directories or directories and files should be mapped
4. Check “Open Report When Created” if you want your browser to display the report after it has been generated
5. Click the “Map” button to start the report generation. Please note that while the Application Progress will show the major stages of the report generation, depending on the number of files and folders it may take a few minutes per stage.

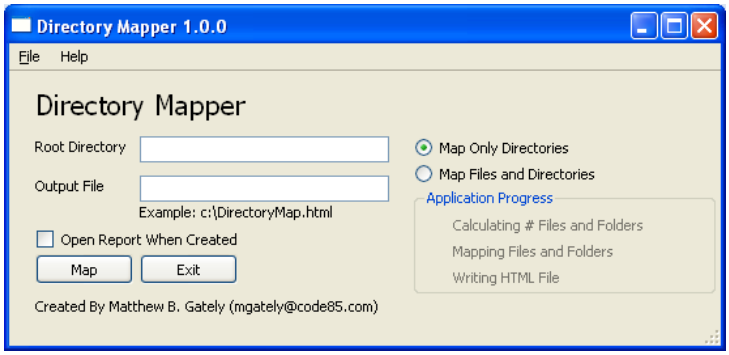

Figure 1: Directory Mapper Application v1.0.0

Important Notes

Folders with large numbers of files and directories may take several minutes to process. Please be patient and do not attempt to close the application until it has finished processing.

Do to the nature of the Windows file system, the calculation of the number of files and folders takes the same amount of time if only directories, or directories and files are being mapped.

Source Code

directorymapper.h

```
#ifndef DIRECTORYMAPPER_H
#define DIRECTORYMAPPER_H

#include <QtGui/QMainWindow>
#include "ui_directorymapper.h"

class DirectoryMapper : public QMainWindow
{
  Q_OBJECT

public:
  DirectoryMapper(QWidget *parent = 0);
  ~DirectoryMapper();

private slots:
        //void browseForFile();
        void mapDirectories();
        void aboutDialog();

private:
  Ui::DirectoryMapperClass ui;
};

#endif // DIRECTORYMAPPER_H
```

directorymapper.cpp

```
#include "directorymapper.h"
#include "functions.h"
#include "fileIO.h"
#include <iostream>
#include <QtGUI>

using namespace std;

string APP_VERSION = "0.1.0";

DirectoryMapper::DirectoryMapper(QWidget *parent)
  : QMainWindow(parent)
{
        ui.setupUi(this);

        // Update User Interface
        string title = "Directory Mapper " + APP_VERSION;
        this->setWindowTitle(title.c_str());
        ui.authorLabel->setText("Created By Matthew B. Gately (mgately@code85.com)");

        // Connect Signals and Slots
        connect(ui.exitButton, SIGNAL(clicked()), this, SLOT(close()));
        connect(ui.mapButton, SIGNAL(clicked()), this, SLOT(mapDirectories()));
        //connect(ui.make1FileButton, SIGNAL(clicked()), this, SLOT(make1File()));

        // Menu Bar Signals and Slots
        connect(ui.actionExit, SIGNAL(triggered()), this, SLOT(close())); // File->Exit
        connect(ui.actionAbout, SIGNAL(triggered()), this, SLOT(aboutDialog())); // Help->About

        // Set Status Tips - These Show In The Status Bar When Mouse Is Hovered
        ui.actionExit->setStatusTip(tr("Exit This Application")); // File->Exit

        ui.directoryOnlyRadioButton->setChecked(true);
```

```
	// Update Application Progress Display
	ui.appProgress1->setEnabled(false);
	ui.appProgress2->setEnabled(false);
	ui.appProgress3->setEnabled(false);

	ui.appProgress1->setText("Calculating # Files and Folders");
	ui.appProgress2->setText("Mapping Files and Folders");
	ui.appProgress3->setText("Writing HTML File");
}

DirectoryMapper::~DirectoryMapper()
{

}

void DirectoryMapper::aboutDialog(void) {
	// Create The About Dialog Text
	string aboutDialogText = "<h1>Directory Mapper " + APP_VERSION + "</h1>";
	aboutDialogText += "<p>This program indexes the selected directory and creates";
	aboutDialogText += " an HTML file displaying all folders and files.</p>";
	aboutDialogText += "<p>Created By Matthew B. Gately (mgately@code85.com)</p>";

	// Display The About Dialog
	QMessageBox::about(this, tr("About Dialog"), aboutDialogText.c_str());
}

void DirectoryMapper::mapDirectories(void) {
	string htmlFile = "";
	string filePath = "";
	string outputFilePath = "";
	QString qFilePath = "";
	int mode, numFiles, numFolders;
	numFiles = 0;
	numFolders = 0;

	htmlFile = "<HTML>";

	// Reset Application Progress
	ui.appProgress1->setEnabled(false);
	ui.appProgress2->setEnabled(false);
	ui.appProgress3->setEnabled(false);

	ui.appProgress1->setText("Calculating # Files and Folders");
	ui.appProgress2->setText("Mapping Files and Folders");
	ui.appProgress3->setText("Writing HTML File");
	qApp->processEvents();

	ui.appProgress1->setEnabled(true);
	qApp->processEvents();

	// Get File Path
	qFilePath = ui.rootDirectoryLineEdit->text();
	QStringToString(qFilePath, filePath);

	// Get Output File Path
	qFilePath = ui.outputFileLineEdit->text();
	QStringToString(qFilePath, outputFilePath);

	// Verify That Path Exists
	// If Path Does Not Exist - Warn User Exit Function
	if (isDirectory(filePath.c_str()) == false) {
		// Message Box Warning
		QMessageBox::about(this, tr("File Does Not Exist"), "Warning - Root Directory Folder Does Not Exist");
		// Exit Function
		return;
	}
```

```
	if (outputFilePath == "") {
		QMessageBox::about(this, tr("Output File Not Specified"), "Warning - Output File Path Not Specified");
	}


	// Get The Number Of Files and Folders
	getNumFilesAndFolders(filePath.c_str(), numFiles, numFolders);
	ui.appProgress1->setText("<font color=green>Number Files and Folders Calculated</font>");
	qApp->processEvents();

	htmlFile += "<h1>Directory Information</h1>";
	htmlFile += "<b>Number of Folders: </b>" + integerToString(numFolders) + "<br>";
	htmlFile += "<b>Number of Files: </b>" + integerToString(numFiles) + "<br>";


	ui.appProgress2->setEnabled(true);
	qApp->processEvents();
	// Determine If You Should Map Just Folders or Files too
	if (ui.directoryOnlyRadioButton->isChecked() == true) {
		mode = 0;
	} else {
		mode = 1;
	}

	// Write Directory Map Header To HTML
	if (mode == 0) {
		htmlFile += "<h1>Directory Map</h1>";
	} else {
		htmlFile += "<h1>Directory and File Map</h1>";
	}

	mapFilesAndFolders(filePath.c_str(), mode, htmlFile);
	ui.appProgress2->setText("<font color=green>Directory Information Mapped</font>");
	qApp->processEvents();

	ui.appProgress3->setEnabled(true);
	qApp->processEvents();
	htmlFile += "</HTML>";
	writeBinaryFile(outputFilePath.c_str(), htmlFile);
	ui.appProgress3->setText("<font color=green>HTML Written</font>");
	qApp->processEvents();

	// If Requested Open Report After Generation
	if (ui.openCheckBox->isChecked() == true) {
		string openOutputFileCommand = "explorer " + outputFilePath;
		system(openOutputFileCommand.c_str());
	}

}
```

Functions Header File

```
#ifndef FUNCTIONS_H_
#define FUNCTIONS_H_

#include <string>
#include <QtGui/QApplication>

using namespace std;

// Function Prototypes
void QStringToString(QString original, string &final);
int StringToInteger(string original);
int charToInt(char character);
```

```
string integerToString(int value);
string toHex(unsigned char input);

#endif /*FUNCTIONS_H_*/
```

Functions Source File

```
#include "functions.h"
#include <sstream> //  Needed For Integer To String Conversion
#include <string>
#include <QtGui/QApplication>
#include <cmath>

using namespace std;

void QStringToString(QString original, string &final) {
	final = "";
	for (int counter = 0; counter < original.length(); counter++) {
		final += original[counter].toAscii();
	}
}

int StringToInteger(string original) {
	int size = original.size();
	int value = 0;
	for (int counter = 0; counter < size; counter++) {
		value += charToInt(original[counter]) * int(pow(double(10), double(size - counter -1)));
	}
	return value;
}

int charToInt(char character) {
	switch (character) {
	case '0':
		return 0;
	case '1':
		return 1;
	case '2':
		return 2;
	case '3':
		return 3;
	case '4':
		return 4;
	case '5':
		return 5;
	case '6':
		return 6;
	case '7':
		return 7;
	case '8':
		return 8;
	case '9':
		return 9;
	default:
		return -1;
	}
	return -1;
}

string integerToString(int value) {
	ostringstream o;
	o << value;
	return o.str();
}

string toHex(unsigned char input) {
```

```
	string output;
	char buffer[2]="";
	char *pbuffer = buffer;
	itoa (input, pbuffer,16);
	output = buffer;

	// itoa function will not put a leading 0
	// if the hex code starts with a zero
	// Check to see if a digit is missing, if so add a 0
	if (output.length() == 1)
		output = "0" + output;

	return output;

}

string intToHex(unsigned int input) {
	string output;
	char buffer[16]="";
	char *pbuffer = buffer;
	itoa (input, pbuffer,16);
	output = buffer;

	return output;
}
```

fileIO.h

```
#ifndef FILEIO_H_
#define FILEIO_H_

#include <string.h>

using namespace std;

double returnFileSize(FILE *myFile);
bool readBinaryFile(string filePath, string &fileContents);
bool writeBinaryFile(string filePath, string fileContents);
bool writeTextFile(string filePath, string fileContents);
bool readTextFile(string filePath, string &fileContents);
int countNumFiles(string path);
bool getNumFilesAndFolders(string path, int &numFilesReturned, int &numFoldersReturned);
bool mapFilesAndFolders(string path, int mode, string &dataContents);
bool isDirectory(string path);

#endif /*FILEIO_H_*/
```

fileIO.cpp

```
#include <iostream>
#include <string.h>
#include <windows.h>
#include <fstream>
#include <io.h>   // For access().
#include <sys/types.h>  // For stat().
#include <sys/stat.h>   // For stat().


#include "fileIO.h"

// Global Constants
bool DEBUG_MODE = false;
```

```
// Global Variables

double returnFileSize(FILE *fileHandle) {
	double start_location = ftell(fileHandle);
	if (fileHandle == NULL)
		return -1;
	fseek(fileHandle, 0, SEEK_END);
	double file_size = ftell(fileHandle);
	fseek(fileHandle, start_location, SEEK_SET);
	return file_size;
}

/*-------------------------------------*/
// Write Binary File Function		//
/*-------------------------------------*/
bool writeBinaryFile(string filePath, string fileContents) {
	double fileSize = fileContents.length();
	char *outputBuffer;

	// Open File For Writing and Get File Handle
	FILE * fileHandle;
	fileHandle = fopen(filePath.c_str(), "wb");

	// If File Opened Properly Write Data To It
	if (fileHandle != NULL) {
		// Create The Output Buffer
		outputBuffer = new char [int(fileSize)];

		// Copy File Contents To Output Buffer
		for (int counter = 0; counter < fileSize; counter++) {
			outputBuffer[counter] = fileContents[counter];
		}

		// Write Output Buffer To File
		fwrite(outputBuffer, 1, fileSize, fileHandle);

		// Delete The Output Buffer From Memory
		delete[] outputBuffer;
	} else {
		fclose(fileHandle);
		return false;
	}

	// Close File
	fclose(fileHandle);

	return true;
}


/*-------------------------------------*/
// Read Binary File Function		//
/*-------------------------------------*/
bool readBinaryFile(string filePath, string &fileContents) {
	double fileSize;
	char *inputBuffer;
	fileContents = "";

	FILE *fileHandle;
	fileHandle = fopen(filePath.c_str(), "rb");
	if (fileHandle != NULL) {
		// Get The File Size
		fileSize = returnFileSize(fileHandle);
		inputBuffer = new char [int(fileSize)];

		fread(inputBuffer, 1, fileSize, fileHandle);
```

```
                        // Copy Buffer To String
                        for (int counter = 0; counter < fileSize; counter++) {
                                fileContents += inputBuffer[counter];
                        }

                        delete[] inputBuffer;

                } else {
                        fclose(fileHandle);
                        return false;
                }
                fclose(fileHandle);
                return true;

}

bool writeTextFile(string filePath, string fileContents) {
                // This Function Takes A Text String and Writes It To A File

                ofstream fileHandle(filePath.c_str());
                if (fileHandle.is_open() == true) {
                        fileHandle << fileContents.c_str();
                        fileHandle.close();
                } else {
                        return false;
                }
                return true;

}

bool readTextFile(string filePath, string &fileContents) {
                // This Function Reads A File Into A String
                string line = "";
                fileContents = "";

                ifstream fileHandle(filePath.c_str());
                if (fileHandle.is_open() == true) {
                        while (!fileHandle.eof()) {
                                getline(fileHandle, line);
                                fileContents += line;
                                if (!fileHandle.eof())
                                        fileContents += "\n";
                        }
                        fileHandle.close();
                } else {
                        fileContents = "";
                        return false;
                }

                return true;
}

bool isDirectory(string path) {
   if ( access(path.c_str(), 0 ) == 0 ) {
      struct stat status;
      stat( path.c_str(), &status );

      if ( status.st_mode & S_IFDIR ) {
                if (DEBUG_MODE)
                        cout << "DEBUG STATEMENT: The directory exists." << endl;
         return true;
      } else {
                if (DEBUG_MODE)
                        cout << "DEBUG STATEMENT: The path you entered is a file." << endl;
      }
```

```
	} else {
		if (DEBUG_MODE)
			cout << "DEBUG STATEMENT: Path doesn't exist." << endl;
	}

	return false;
}

// Mode is set to 0 for folders only and 1 for files too
bool mapFilesAndFolders(string path, int mode, string &dataContents) {
		dataContents += "<ul>";
		// Variable Declarations
			WIN32_FIND_DATA FindFileData;
			HANDLE hFind = INVALID_HANDLE_VALUE;

			int numFiles = 0;
			int numFolders = 0;

			char directoryPath[MAX_PATH];

			for(int counter = 0; counter < path.length(); counter++) {
				directoryPath[counter] = path.c_str()[counter];
			}
			directoryPath[path.length()] = '\0'; // Null Terminator After File Path

			if (DEBUG_MODE)
				cout << "DEBUG STATEMENT: The Current Path Is: " << directoryPath << endl;

			WCHAR wdPath[MAX_PATH];
			strncat(directoryPath, "\\*", 3);

			for (int counter = 0; counter < MAX_PATH; counter++) {
				wdPath[counter] = directoryPath[counter];
			}

			hFind = FindFirstFile(wdPath, &FindFileData);

			if (hFind == INVALID_HANDLE_VALUE)
				cout << "ERROR: Invalid File Path In Count Num Files and Folders Function" << endl;

			string tempPath = "";
			while (FindNextFile(hFind, &FindFileData) != 0) {
				tempPath = "";
				for (int counter = 0; counter < MAX_PATH; counter++) {
					if (char(FindFileData.cFileName[counter]) == '\0')
						break;
					else {
						//cout << char(FindFileData.cFileName[counter]);
						tempPath += char(FindFileData.cFileName[counter]);
					}
				}
				//cout << endl;
				string tempPath2 = tempPath;
				tempPath = path + "\\" + tempPath;

				// Verify The File Is Not Telling Itself To Go Up 1 Level
				if (isDirectory(tempPath) == true && tempPath2 != "..") {
					numFolders++;
					int tempNumA = 0;
					int tempNumB = 0;
					dataContents += "<b><li>" + tempPath2 + "</li></b>";
					mapFilesAndFolders(tempPath, mode, dataContents);
					numFiles += tempNumA;
					numFolders += tempNumB;

				} else {
```

```
                                        if (tempPath2 != "..") {

                                                if (mode == 1) {
                                                        dataContents += "<li>" + tempPath2 + "</li>";
                                                }
                                                numFiles++;
                                        }
                                }
                        }

                        FindClose(hFind);
                        if (DEBUG_MODE == true) {
                                cout << "The Current Number Of Files Is: " << numFiles << endl;
                                cout << "The Current Number Of Folders Is: " << numFolders << endl;
                        }

                        dataContents += "</ul>";

                        return true;


}


bool getNumFilesAndFolders(string path, int &numFilesReturned, int &numFoldersReturned) {
        // Variable Declarations
        WIN32_FIND_DATA FindFileData;
        HANDLE hFind = INVALID_HANDLE_VALUE;

        int numFiles = 0;
        int numFolders = 0;

        char directoryPath[MAX_PATH];

        for(int counter = 0; counter < path.length(); counter++) {
                directoryPath[counter] = path.c_str()[counter];
        }
        directoryPath[path.length()] = '\0'; // Null Terminator After File Path

        if (DEBUG_MODE)
                cout << "DEBUG STATEMENT: The Current Path Is: " << directoryPath << endl;

        WCHAR wdPath[MAX_PATH];
        strncat(directoryPath, "\\*", 3);

        for (int counter = 0; counter < MAX_PATH; counter++) {
                wdPath[counter] = directoryPath[counter];
        }

        hFind = FindFirstFile(wdPath, &FindFileData);

        if (hFind == INVALID_HANDLE_VALUE)
                cout << "ERROR: Invalid File Path In Count Num Files and Folders Function" << endl;

        string tempPath = "";
        while (FindNextFile(hFind, &FindFileData) != 0) {
                tempPath = "";
                for (int counter = 0; counter < MAX_PATH; counter++) {
                        if (char(FindFileData.cFileName[counter]) == '\0')
                                break;
                        else {
                                //cout << char(FindFileData.cFileName[counter]);
                                tempPath += char(FindFileData.cFileName[counter]);
                        }
                }
                //cout << endl;
```

```
                        string tempPath2 = tempPath;
                        tempPath = path + "\\" + tempPath;

                        // Verify The File Is Not Telling Itself To Go Up 1 Level
                        if (isDirectory(tempPath) == true && tempPath2 != "..") {
                                numFolders++;
                                int tempNumA = 0;
                                int tempNumB = 0;
                                getNumFilesAndFolders(tempPath, tempNumA, tempNumB);
                                numFiles += tempNumA;
                                numFolders += tempNumB;
                        } else {
                                if (tempPath2 != "..") {
                                        numFiles++;
                                }
                        }
                }

                FindClose(hFind);
                if (DEBUG_MODE == true) {
                        cout << "The Current Number Of Files Is: " << numFiles << endl;
                        cout << "The Current Number Of Folders Is: " << numFolders << endl;
                }
                numFilesReturned = numFiles;
                numFoldersReturned = numFolders;

                return true;

}

int countNumFiles(string path) {
                // Variable Declarations
                WIN32_FIND_DATA FindFileData;
                HANDLE hFind = INVALID_HANDLE_VALUE;
                int numFiles = 0;
                int lastLoc = 0;

                char directoryPath[MAX_PATH];

                for (int counter = 0; counter < path.length(); counter++) {
                        directoryPath[counter] = path.c_str()[counter];
                        lastLoc = counter;
                }
                lastLoc++;
                directoryPath[lastLoc] = '\0';

                if (DEBUG_MODE)
                        cout << "DEBUG STATEMENT: The Current Path Is: " << directoryPath << endl;
                WCHAR wdPath[MAX_PATH];

                strncat(directoryPath, "\\*", 3);

                for (int counter = 0; counter < MAX_PATH; counter++) {
                        wdPath[counter] = directoryPath[counter];
                }

                hFind = FindFirstFile(wdPath, &FindFileData);

                if (hFind == INVALID_HANDLE_VALUE) {
                        cout << "Error: Invalid File Path" << endl;
                }

                //cout << FindFileData.cFileName << endl;
                string tempPath = "";
                while (FindNextFile(hFind, &FindFileData) != 0) {
                        tempPath = "";
```

```
                for (int counter = 0; counter < MAX_PATH; counter++) {
                        if (char(FindFileData.cFileName[counter]) == '\0')
                                break;
                        else {
                                //cout << char(FindFileData.cFileName[counter]);
                                tempPath += char(FindFileData.cFileName[counter]);
                        }
                }
                //cout << endl;
                string tempPath2 = tempPath;
                tempPath = path + "\\" + tempPath;

                // Verify The File Is Not Telling Itself To Go Up 1 Level
                if (isDirectory(tempPath) == true && tempPath2 != "..") {
                        numFiles += countNumFiles(tempPath);
                } else {
                        if (tempPath2 != "..") {
                                numFiles++;
                        }
                }
        }

        FindClose(hFind);

        return numFiles;
}
```

Revision History

Version 0.1.0

Release Date: 08/30/2008
Build Type: Initial Working Debug Build
Changes: None

Version 1.0.0

Release Date: 09/01/2008
Build Type: Release Build
Changes:

Application compiled in release mode to reduce file and dll size