ScreenRecorder: A Utility for Creating Screenshot Video Using Only Original Equipment Manufacturer (OEM) Software on Microsoft Windows Systems

by Mary K Arthur

Approved for public release; distribution is unlimited.
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
ScreenRecorder: A Utility for Creating Screenshot Video Using Only Original Equipment Manufacturer (OEM) Software on Microsoft Windows Systems

Mary K Arthur
Weapons and Materials Research Directorate, ARL

Approved for public release; distribution is unlimited.
**REPORT DOCUMENTATION PAGE**

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

**PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

<table>
<thead>
<tr>
<th>1. REPORT DATE (DD-MM-YYYY)</th>
<th>2. REPORT TYPE</th>
<th>3. DATES COVERED (From - To)</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2015</td>
<td>Final</td>
<td>August 2014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE AND SUBTITLE</th>
<th>5a. CONTRACT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScreenRecorder: A Utility for Creating Screenshot Video Using Only Original Equipment Manufacturer (OEM) Software on Microsoft Windows Systems</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5b. GRANT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5c. PROGRAM ELEMENT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. AUTHOR(S)</th>
<th>5d. PROJECT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary K Arthur</td>
<td>AH80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5e. TASK NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5f. WORK UNIT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</th>
<th>8. PERFORMING ORGANIZATION REPORT NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>US Army Research Laboratory</td>
<td>ARL-TN-0658</td>
</tr>
<tr>
<td>ATTN: RDRL-WML-A</td>
<td></td>
</tr>
<tr>
<td>Aberdeen Proving Ground, MD 21005-5069</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10. SPONSOR/MONITOR'S ACRONYM(S)</th>
<th>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>12. DISTRIBUTION/AVAILABILITY STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved for public release; distribution is unlimited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>14. ABSTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being able to save screenshots directly to a file and to record activity on a desktop or within a specific window is extremely useful for a variety of reasons. Although third-party screen-recording software is readily available online from a variety of sources, such software often presents a security concern and would require administrative permissions to install. This report presents a software utility for capturing series of desktop or window screenshots using only original equipment manufacturer software on systems running Microsoft Windows.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. SUBJECT TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenshot, screen capture, screen record, desktop, window, handle, Microsoft, Windows, original equipment manufacturer, OEM, Visual Studio, Movie Maker, C++, object-oriented</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16. SECURITY CLASSIFICATION OF:</th>
<th>17. LIMITATION OF ABSTRACT</th>
<th>18. NUMBER OF PAGES</th>
<th>19a. NAME OF RESPONSIBLE PERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. REPORT</td>
<td>Unclassified</td>
<td></td>
<td>Mary K Arthur</td>
</tr>
<tr>
<td>b. ABSTRACT</td>
<td>Unclassified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c. THIS PAGE</td>
<td>Unclassified</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UU</td>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19b. TELEPHONE NUMBER (Include area code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>410-278-6110</td>
</tr>
</tbody>
</table>

**Standard Form 298 (Rev. 8/98)**

Prescribed by ANSI Std. Z39.18
Contents

List of Figures iv
Acknowledgments v

1. Introduction 1

2. ScreenRecorder: Static Member Functions, Screen Capture 2
   2.1 CaptureDesktop ................................................................. 2
   2.2 CaptureByTitle ................................................................. 3
   2.3 CaptureByHandle ............................................................... 5

3. ScreenRecorder: Class Variables, Constructors, Accessors, and Mutators 6
   3.1 Constructors ......................................................................... 6
   3.2 Accessors ............................................................................ 7
   3.3 Mutators ............................................................................... 7

4. ScreenRecorder: Non-Static Member Functions, Screen Record 8
   4.1 RecordDesktop ........................................................................ 8
   4.2 RecordByTitle ......................................................................... 9
   4.3 RecordByHandle ..................................................................... 11

5. Conclusion 12

6. References and Notes 13

Appendix A. ScreenRecorderExamples_Capture.cc 15
Appendix B. ScreenRecorderExamples_Record.cc 19
Appendix C. ScreenRecorder.h 23
Appendix D. ScreenRecorder.cc 27
Appendix E. MATLAB Script for Creating AVI Movie from Screenshots 35
Distribution List 37
List of Figures

Fig. 1  Static member function (screen capture) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and Appendix D for full class implementation in ScreenRecorder.cc. .........................................................2

Fig. 2  Sample usage of CaptureDesktop functions from
        ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of
        ScreenRecorderExamples_Capture.cc. ........................................................................3

Fig. 3  Sample usage of CaptureByTitle functions from
        ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of
        ScreenRecorderExamples_Capture.cc. ........................................................................4

Fig. 4  Sample usage of CaptureByHandle functions from
        ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of
        ScreenRecorderExamples_Capture.cc. ........................................................................5

Fig. 5  Constructor declarations in ScreenRecorder.h. See Appendix C for full
documentation of ScreenRecorder.h, and appendix D for full class implementation in
        ScreenRecorder.cc. .................................................................................................6

Fig. 6  Accessor declarations in ScreenRecorder.h. See Appendix C for full documentation of
        ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc. .....7

Fig. 7  Mutator declarations in ScreenRecorder.h. See Appendix C for full documentation of
        ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc. ........7

Fig. 8  Non-static member function (screen record) declarations in ScreenRecorder.h. See Appendix C for full
documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc. ........7

Fig. 9  Sample usage of RecordDesktop functions from
        ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of
        ScreenRecorderExamples_Record.cc. ........................................................................9

Fig. 10 Sample usage of RecordByTitle functions from
        ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of
        ScreenRecorderExamples_Record.cc. .......................................................................10

Fig. 11 Sample usage of RecordByHandle functions from
        ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of
        ScreenRecorderExamples_Record.cc. .......................................................................11
Acknowledgments

I would like to thank Ben Breech for his extremely thorough technical review, feedback, and suggestions that greatly improved the quality of the ScreenRecorder utility and this report. I would also like to thank Mark Gatlin for his editorial review.
1. Introduction

Being able to save screenshots directly to a file and to record activity on a desktop or within a specific window is extremely useful for a variety of reasons including, but not limited to, diagnosing computer problems, teaching and/or demonstrating software, and generating videos for presentations. Although third-party screen-recording software is readily available online from a variety of sources, such software often presents a security concern and would require administrative permissions to install. This report presents ScreenRecorder, a software utility for capturing series of desktop or window screenshots using only original equipment manufacturer (OEM) software on systems running Microsoft Windows.¹

The ScreenRecorder utility was developed as an object-oriented C++ class within Microsoft Visual Studio.² It has been tested on and is compatible with Microsoft Vista, 7, and 8 and Visual Studio Express 2008, 2010, and 2012. The utility supports 5 image formats: Windows Bitmap (BMP), Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), and Tagged Image File Format (TIFF). Once the ScreenRecorder utility has saved a series of screenshots in the desired format, Windows Movie Maker³ can be used to combine the screenshot images into a Windows media file (WMV) or Audio-Video Interleaved (AVI) movie file. This report does not explicitly address creating a video from a series of images but rather directs the user to Microsoft’s Windows Movie Maker tutorial⁴ for further information regarding creating videos from images using Microsoft OEM software.⁵

Functions within the ScreenRecorder utility fall into 3 categories: 1) static screen capture member functions for saving a single screenshot to a file that can be invoked without using a ScreenRecorder object, 2) functions for creating and manipulating ScreenRecorder objects (constructors, accessors, and mutators), and 3) non-static screen recording member functions for saving a series of screenshots to file that can only be accessed through a ScreenRecorder object.

The use of the ScreenRecorder utility assumes a basic understanding of compiling and running C++ code within Microsoft Visual Studio. This report does not attempt to explain the inner workings of the ScreenRecorder utility but is published as a means to distribute the source code and act as a user's manual. All source code is provided to the user within the appendices. The source code is completely self-contained, and the user need only include the files in a Visual Studio project and link to the following 3 standard Windows libraries: user32, gdi32, and gdiplus. See example codes in Appendices A and B for an example of how to link these libraries.
2. **ScreenRecorder: Static Member Functions, Screen Capture**

The ScreenRecorder utility includes 4 static member functions for capturing a single screenshot image: CaptureDesktop, CaptureByTitle, and 2 versions of CaptureByHandle (Fig. 1).

```
// **** Static Methods - Screen Capture *************************************/
static bool CaptureDesktop(std::string saveAsFile = "ScreenCapture.png",
                         std::string saveAsPath = "");
static bool CaptureByTitle(std::string title,
                           std::string saveAsFile = "ScreenCapture.png",
                           std::string saveAsPath = "");
static bool CaptureByHandle(std::string handle,
                            std::string saveAsFile = "ScreenCapture.png",
                            std::string saveAsPath = "");
static bool CaptureByHandle(HWND hWnd,
                           std::string saveAsFile = "ScreenCapture.png",
                           std::string saveAsPath = "");
```

Fig. 1 Static member function (screen capture) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and Appendix D for full class implementation in ScreenRecorder.cc.

2.1 **CaptureDesktop**

The CaptureDesktop function saves a single image of the desktop. CaptureDesktop takes up to 2 string parameters.

If no parameters are supplied, the image of the desktop is saved as ScreenCapture.png in the same directory as the executable (Fig. 2, lines 31 and 32).

If a single string parameter is supplied, this parameter specifies the save-as filename. The image of the desktop is then saved with the specified filename in the same directory as the executable (Fig. 2, lines 37 and 38).

If 2 string parameters are supplied, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the desktop is then saved with the specified filename in the specified directory (Fig. 2, lines 43–46).
ScreenRecorder recorder; // ScreenRecorder object

// **** ScreenRecorder::CaptureDesktop **********************************************

// CaptureDesktop
if (!ScreenRecorder::CaptureDesktop() || !recorder.CaptureDesktop()) {
    std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
}

// CaptureDesktop(saveAsFile)
if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.bmp") || !recorder.CaptureDesktop("CaptureDesktop.bmp")) {
    std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
}

// CaptureDesktop(saveAsFile, saveAsPath)
if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.gif", "C:\Documents\ScreenRecorder") || !recorder.CaptureDesktop("CaptureDesktop.gif", "C:\Documents\ScreenRecorder")) {
    std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
}

Fig. 2 Sample usage of CaptureDesktop functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.

The return value of CaptureDesktop indicates whether or not an image of the desktop was successfully saved. If the user specifies the filename, the CaptureDesktop function will return false if the specified filename does not end with one of the supported file format extensions (i.e., bmp, gif, jpg/jpeg, png, or tif/tiff). If the user specifies a save path that does not exist, CaptureDesktop will return false.

Windows is generally case-insensitive; all functions in the ScreenRecorder utility are case-insensitive.

2.2 CaptureByTitle

The CaptureByTitle function saves a single image of the window identified by the specified title. CaptureByTitle requires at least one string parameter (the title of the window to capture) followed by up to 2 string parameters.

If only the required title string parameter is supplied, the image of the specified window is saved as ScreenCapture.png in the same directory as the executable (Fig. 3, lines 53 and 54).
Fig. 3 Sample usage of CaptureByTitle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.

If a single string parameter is supplied in addition to the required title string parameter, this parameter specifies the save-as filename. The image of the specified window is then saved with the specified filename in the same directory as the executable (Fig. 3, lines 59–62).

If 2 string parameters are supplied in addition to the required title string parameter, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the specified window is then saved with the specified filename in the specified directory (Fig. 3, lines 67–72).

The return value of CaptureByTitle indicates whether or not the specified window was found and an image of this window was successfully saved. If the user specifies the filename, the CaptureByTitle function will return false if the specified filename does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, CaptureByTitle will return false.

The “title” that appears at the top of a window (in the title bar) is not necessarily the actual title of the window. The easiest way to determine the actual title of the window is to open Windows Task Manager (ctrl + shift + esc), select the Applications tab, and identify the task that you are attempting to capture. The task names are the actual window titles.
2.3 CaptureByHandle

Both CaptureByHandle functions save a single image of the window identified by the specified window handle. Both CaptureByHandle functions require one parameter specifying the handle to the window to capture followed by up to 2 optional string parameters. The difference between the 2 CaptureByHandle functions is the data type of the required parameter specifying the handle to the window to capture. This parameter can either be a hex string identifier to the handle (e.g., “0012079E”) or a Microsoft Windows window handle (HWND).

If only the required handle identifier parameter is supplied, the image of the specified window is saved as ScreenCapture.png in the same directory as the executable (Fig. 4, lines 85–88).

```cpp
ScreenRecorder recorder; // ScreenRecorder object

// **** ScreenRecorder::CaptureByHandle *********************************************
// Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");

// Hex value of handleObject saved as a string
std::string handleString = toString(handleObject);

// CaptureByHandle(windowHandle)
if (!ScreenRecorder::CaptureByHandle(handleObject) ||
    !recorder.CaptureByHandle(handleObject) ||
    !ScreenRecorder::CaptureByHandle(handleString) ||
    !recorder.CaptureByHandle(handleString)) {
    std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))

// CaptureByHandle(windowHandle, saveAsFile)
if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.tiff") ||
    !recorder.CaptureByHandle(handleObject, "CaptureByHandle.tif") ||
    !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.bmp") ||
    !recorder.CaptureByHandle(handleString, "CaptureByHandle.gif")) {
    std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))

// CaptureByHandle(windowHandle, saveAsFile, saveAsPath)
if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.jpg",
    "C:\Documents\ScreenRecorder") ||
    !recorder.CaptureByHandle(handleObject, "CaptureByHandle.png",
    "C:\Documents\ScreenRecorder") ||
    !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.tif",
    "C:\Documents\ScreenRecorder") ||
    !recorder.CaptureByHandle(handleString, "CaptureByHandle.bmp",
    "C:\Documents\ScreenRecorder")) {
    std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))
```

Fig. 4 Sample usage of CaptureByHandle functions from ScreenRecorderExamples_Capture.cc. See Appendix A for full documentation of ScreenRecorderExamples_Capture.cc.
If a single string parameter is supplied in addition to the required handle identifier parameter, this parameter specifies the save-as filename. The image of the specified window is then saved with the specified filename in the same directory as the executable (Fig. 4, lines 93–96).

If 2 string parameters are supplied in addition to the required handle identifier parameter, the first parameter specifies the save-as filename and the second is the path to the desired save location. The image of the specified window is then saved with the specified filename in the specified directory (Fig. 4, lines 101–108).

The return value of CaptureByHandle indicates whether or not the specified window was found and an image of this window was successfully saved. If the user specifies the filename, the CaptureByHandle function will return false if the filename does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, CaptureByHandle will return false.

3. ScreenRecorder: Class Variables, Constructors, Accessors, and Mutators

A ScreenRecorder object has 3 class variables: record time, frame rate, and time step. The record-time variable is the duration of time in seconds over which the recorder object will capture a series of screenshots; it has type double and default value of 60.0 s. The frame-rate variable is the number of screenshots that will be captured each second; it has type double and default value of 1.0 frame per second (fps). The time-step variable is the time between screenshots in seconds; it has type double and default value of 1.0 s. The frame rate and time step are inversely proportional and related by the following equation:

\[
\text{timeStep} = \frac{1}{\text{frameRate}}.
\]

The ScreenRecorder utility includes 3 constructors, 3 accessor functions, and 3 mutator functions.

3.1 Constructors

The default constructor for a ScreenRecorder object does not take any parameters and sets the class variables to their default values (Fig. 5, line 37).

```cpp
// **** Constructors #####################################################################################################
ScreenRecorder();
ScreenRecorder(double recordTime, int frameRate);
ScreenRecorder(double recordTime, double timeStep);
```

Fig. 5 Constructor declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h, and appendix D for full class implementation in ScreenRecorder.cc.
The remaining 2 constructors both require 2 parameters. The first parameter is the same for both constructors, has type double, and specifies the record time in seconds. The second parameter either has type integer and specifies the frame rate\textsuperscript{9} in frames per second (figure 5, line 38) or has type double and specifies the time step in seconds (figure 5, line 39). If a record time of less than 0.0 is specified, an error message is printed to the standard error stream (std::cerr) and the record time is set to the default 60.0 s. If a frame rate of 0 or less\textsuperscript{10} is specified, an error message is printed to std::cerr and the frame rate is set to the default 1.0 fps. If a time step of 0.0 or less is specified, an error message is printed to std::cerr and the time step is set to the default 1.0 s.

3.2 Accessors

Accessor functions, often called “getters”, are methods that return the state (value) of private member variables. The ScreenRecorder has an accessor for each of its 3 private member variables: GetRecordTime, GetFrameRate, and GetTimeStep (Fig. 6). Accessors do not take any parameters. GetRecordTime returns the duration of time in seconds over which the recorder object will capture a series of screenshots. GetFrameRate returns the frame rate of the recorder object in frames per second. GetTimeStep returns the time step of the recorder object in seconds.

```cpp
41 // **** Accessors ..............................................................................................
42 double GetRecordTime();
43 double GetFrameRate();
44 double GetTimeStep();
```

Fig. 6 Accessor declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.

3.3 Mutators

Mutator functions, often called “setters”, are methods for changing the state (value) of private member variables. The ScreenRecorder has a mutator for each of its 3 private member variables: SetRecordTime, SetFrameRate, and SetTimeStep (Fig. 7).

```cpp
47 // **** Mutators ..............................................................................................
48 void SetRecordTime(double recordTime);
49 void SetFrameRate(double frameRate);
50 void SetTimeStep(double dT);
```

Fig. 7 Mutator declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.

SetRecordTime takes one parameter of type double and with units in seconds and sets the duration of time over which the recorder object will capture a series of screenshots to this value. If a parameter value of less than 0.0 is passed to SetRecordTime, an error message is printed to std::cerr and the record time is set to the default 60.0 s.

SetFrameRate takes one parameter of type double and with units in frames per second and sets the frame rate of the recorder object to this value. If a parameter value that is less than or equal
to 0.0 is passed to SetFrameRate, an error message is printed to std::cerr and the frame rate is set to the default 1.0 fps. SetFrameRate also changes the object’s time step to be equal to 1.0 divided by the new frame rate.

SetTimeStep takes one parameter of type double and with units in seconds and sets the time step of the recorder object to this value. If a parameter value that is less than or equal to 0.0 is passed to SetTimeStep, an error message is printed to std::cerr and the time step is set to the default 1.0 s. SetTimeStep also changes the object’s frame rate to be equal to 1.0 divided by the new time step.

4. ScreenRecorder: Non-Static Member Functions, Screen Record

The ScreenRecorder utility includes 4 non-static member functions for recording a series of screenshot images: RecordDesktop, RecordByTitle, and 2 versions of RecordByHandle (Fig. 8).

```cpp
bool RecordDesktop(std::string fileNameBase = "ScreenRecorder.png",
                   std::string saveAsPath = "");
bool RecordByTitle(std::string title,
                   std::string fileNameBase = "ScreenRecorder.png",
                   std::string saveAsPath = "");
bool RecordByHandle(HWND hWnd,
                    std::string fileNameBase = "ScreenRecorder.png",
                    std::string saveAsPath = "");
bool RecordByHandle(std::string handle,
                    std::string fileNameBase = "ScreenRecorder.png",
                    std::string saveAsPath = "");
```

Fig. 8 Non-static member function (screen record) declarations in ScreenRecorder.h. See Appendix C for full documentation of ScreenRecorder.h and appendix D for full class implementation in ScreenRecorder.cc.

4.1 RecordDesktop

The RecordDesktop function saves a series of images of the desktop. RecordDesktop takes up to 2 string parameters.

If no parameters are supplied, a ScreenRecorder directory is created, if it does not already exist, in the same directory as the executable. The images of the desktop are saved as N_ScreenRecorder.png in this directory where N is the image number starting from 0 (Fig. 9, line 68).
// **** ScreenRecorder::RecordDesktop ***********************************************
//
// RecordDesktop
if (!recorder1.RecordDesktop()) {
  std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
} // if (!recorder1.RecordDesktop())

// RecordDesktop(saveAsFile)
if (!recorder2.RecordDesktop("RecordDesktop.bmp")) {
  std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
} // if (!recorder2.RecordDesktop(...))

// RecordDesktop(saveAsFile, saveAsPath)
if (!recorder3.RecordDesktop("RecordDesktop.gif",
  "C:\Documents\ScreenRecorder") ) {
  std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
} // if (!recorder3.RecordDesktop(...))

Fig. 9 Sample usage of RecordDesktop functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.

If a single string parameter is supplied, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the desktop are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 9, line 73).

If 2 string parameters are supplied, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the desktop are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 9, lines 78 and 79).

The return value of RecordDesktop indicates whether or not the desired series of images of the desktop were successfully saved. If the user specifies the filename base, the RecordDesktop function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordDesktop will return false.

4.2 RecordByTitle
The RecordByTitle function saves a series of images of the window identified by the specified title. RecordByTitle requires at least one string parameter (the title of the window to capture) followed by up to 3 string parameters.

If only the required title string parameter is supplied, a ScreenRecorder directory is created, if it does not already exist, in the same directory as the executable. The images of the specified window are then saved as N_ScreenRecorder.png in this directory where N is the image number starting from 0 (Fig. 10, line 86).
If a single string parameter is supplied in addition to the required title string parameter, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0, and filenameBase is the user-specified filename base (Fig. 10, lines 91–92).

If 2 string parameters are supplied in addition to the required title string parameter, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0, and filenameBase is the user-specified filename base (Fig. 10, lines 97–99).

Fig. 10  Sample usage of RecordByTitle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.

The return value of RecordByTitle indicates whether or not the specified window was found and the desired series of images of this window were successfully saved. If the user specifies the filename base, the RecordByTitle function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordByTitle will return false.

As in CaptureByTitle, the “title” that appears at the top of a window (in the title bar) is not necessarily the actual title of the window. The easiest way to determine the actual title of the window is to open Windows Task Manager (ctrl + shift + esc), select the Applications tab, and identify the task that you are attempting to record. The task names are the actual window titles.
4.3 RecordByHandle

Both RecordByHandle functions save a series of images of the window identified by the specified window handle. Both RecordByHandle functions require one parameter specifying the handle to the window to record followed by up to 2 optional string parameters. The difference between the 2 RecordByHandle functions is the data type of the required parameter specifying the handle to the window to record. This parameter can either be a hex string identifier to the handle (e.g., “0012079E”) or an HWND.

If only the required handle identifier parameter is supplied, a ScreenRecorder directory is created, if it does not already exist, in the same directory as the executable. The images of the specified window are then saved as N_ScreenRecorder.png in this directory where N is the image number starting from 0 (Fig. 11, lines 112 and 113).

If a single string parameter is supplied in addition to the required handle identifier parameter, this parameter specifies the base of the save-as filename. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the same directory as the executable. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 11, lines 118 and 119).

```cpp
103 // **** ScreenRecorder::RecordByHandle **********************************************
104 // Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
105 HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");
106
107 // Hex value of handleObject saved as a string
108 std::string handleString = toString(handleObject);
109
110 // RecordByHandle(windowHandle)
111 if (!recorder1.RecordByHandle(handleObject) ||
112     !recorder1.RecordByHandle(handleString)) {
113     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
114 } // if (!recorder1.RecordByHandle(...))
115
116 // RecordByHandle(windowHandle, saveAsFile)
117 if (!recorder2.RecordByHandle(handleObject, "RecordByHandle.tif") ||
118     !recorder2.RecordByHandle(handleString, "RecordByHandle.bmp")) {
119     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
120 } // if (!recorder2.RecordByHandle(...))
121
122 // RecordByHandle(windowHandle, saveAsFile, saveAsPath)
123 if (!recorder3.RecordByHandle(handleObject, "RecordByHandle.gif",
124     "C:\Documents\ScreenRecorder") ||
125     !recorder3.RecordByHandle(handleString, "RecordByHandle.jpeg",
126     "C:\Documents\ScreenRecorder")) {
127     std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
128 } // if (!recorder3.RecordByHandle(...))
```

Fig. 11 Sample usage of RecordByHandle functions from ScreenRecorderExamples_Record.cc. See Appendix B for full documentation of ScreenRecorderExamples_Record.cc.
If 2 string parameters are supplied in addition to the required handle identifier parameter, the first parameter specifies the base of the save-as filename and the second is the path to the desired save location. A directory with the same name as the filename base with the extension removed is then created, if it does not already exist, in the specified directory. The images of the specified window are saved as N_filenameBase in this directory. Again, N is the image number starting from 0 and filenameBase is the user-specified filename base (Fig. 11, lines 124–127).

The return value of RecordByHandle indicates whether or not the specified window was found and the desired series of images of this window were successfully saved. If the user specifies the filename base, the RecordByHandle function will return false if the filename base does not end with one of the supported file format extensions. If the user specifies a save path that does not exist, RecordByHandle will return false.

5. Conclusion

The ScreenRecorder utility was developed for saving series of desktop or window screenshots directly to file on any system running Microsoft Windows as an alternative to restrictive third-party screen recording software options. With the use of Microsoft Visual Studio, the ScreenRecorder utility was developed as a C++ class that can be compiled as a library (static or dynamic) to be linked into a project or can easily be compiled into an executable with the use of wrapper code. One major advantage of structuring the ScreenRecorder utility this way and distributing the source code (as opposed to only distributing an executable) is that the ScreenRecorder can be directly embedded into and manipulated within new and existing models.

The ScreenRecorder utility has been tested on and is compatible with Microsoft Vista, 7, and 8 and Visual Studio Express 2008, 2010, and 2012. The utility supports BMP, GIF, JPEG, PNG, and TIFF image formats. Once the ScreenRecorder utility has saved a series of screenshots in the desired format, Windows Movie Maker (or other appropriate software) can be used to combine the screenshot images into a WMV or AVI movie file.

Examples of wrapper codes and linking options are provided to the user in Appendices A and B. All source code is provided to the user in Appendices C and D.
6. References and Notes


5. Although it is not OEM software, MATLAB provides another easy way to combine the screenshots into an AVI movie. See Appendix E for more details.


7. In computing, a handle is a data type that represents and provides access to a resource loaded in memory.


9. The frame rate in this constructor is specified as an integer, but the frame rate of a ScreenRecorder object will have type double. This was done to allow for 2 overloaded constructors, in which case both constructors could not take the same sequence of parameter types [i.e., ScreenRecorder(double, double)]. This restricts the value of the frame rate when setting it through the constructor. If a noninteger frame rate is desired, the user can either calculate the corresponding time step and use the appropriate constructor or set the frame rate using the SetFrameRate mutator discussed in Section 3.3.

10. When error checking for equality in parameter values, the ScreenRecorder utility uses a tolerance of 1.0e-10.
INTENTIONALLY LEFT BLANK.
Appendix A. ScreenRecorderExamples_Capture.cc

This appendix appears in its original form, without editorial change.
Examples of using screen capture functions.

```cpp
#include <stdio>
#include <iostream>
#include <sstream>
#include "ScreenRecorder.h"

// Link user32, gdi32, and gdiplus libraries
#pragma comment(lib, "user32")
#pragma comment(lib, "gdi32")
#pragma comment(lib, "gdiplus")

template <typename T >
std::string toString(T x) {
    // Converts x to a std::string
    std::stringstream sstr;
    sstr << x;
    return sstr.str();
}

int main(int argc, char* argv[]) {
    ScreenRecorder recorder; // ScreenRecorder object

    // **** ScreenRecorder::CaptureDesktop *******************************
    if (!ScreenRecorder::CaptureDesktop() || !recorder.CaptureDesktop()) {
        std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
    }

    // ScreenRecorder::CaptureDesktop(saveAsFile)
    if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.bmp") ||
        !recorder.CaptureDesktop("CaptureDesktop.bmp")) {
        std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
    }

    // ScreenRecorder::CaptureDesktop(saveAsFile, saveAsPath)
    if (!ScreenRecorder::CaptureDesktop("CaptureDesktop.gif",
        "C:\Documents\ScreenRecorder") ||
        !recorder.CaptureDesktop("CaptureDesktop.gif",
        "C:\Documents\ScreenRecorder")) {
        std::cerr << "ScreenRecorder::CaptureDesktop failed" << std::endl;
    }

    // **** ScreenRecorder::CaptureByTitle *******************************
    if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
        std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
    }

    // ScreenRecorder::CaptureByTitle(saveAsFile)
    if (!ScreenRecorder::CaptureByTitle("CaptureByTitle.gif",
        "C:\Documents\ScreenRecorder") {
        std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
    }
```
if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
    "CaptureByTitle.png",
    "C:\\Documents\\ScreenRecorder")
||
    !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
    "CaptureByTitle.png",
    "C:\\Documents\\ScreenRecorder")) {
  std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
} // if (!CaptureByTitle(...))

// CaptureByTitle(windowTitle, saveAsFile, saveAsPath)
if (!ScreenRecorder::CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
    "CaptureByTitle.png",
    "C:\\Documents\\ScreenRecorder")
||
    !recorder.CaptureByTitle("Microsoft PowerPoint - [Sample.pptx]",
    "CaptureByTitle.png",
    "C:\\Documents\\ScreenRecorder")) {
  std::cerr << "ScreenRecorder::CaptureByTitle failed" << std::endl;
} // if (!CaptureByTitle(...))

// **** ScreenRecorder::CaptureByHandle *********************************************
// Handle object to window titled "Microsoft PowerPoint - [Sample.pptx]"
HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");

// Hex value of handleObject saved as a string
std::string handleString = toString(handleObject);

// CaptureByHandle(windowHandle)
if (!ScreenRecorder::CaptureByHandle(handleObject)
||
    !recorder.CaptureByHandle(handleObject)
||
    !ScreenRecorder::CaptureByHandle(handleString)
||
    !recorder.CaptureByHandle(handleString)) {
  std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))

// CaptureByHandle(windowHandle, saveAsFile)
if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.tif")
||
    !recorder.CaptureByHandle(handleObject, "CaptureByHandle.tif")
||
    !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.bmp")
||
    !recorder.CaptureByHandle(handleString, "CaptureByHandle.gif")) {
  std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))

// CaptureByHandle(windowHandle, saveAsFile, saveAsPath)
if (!ScreenRecorder::CaptureByHandle(handleObject, "CaptureByHandle.jpg",
    "C:\\Documents\\ScreenRecorder")
||
    !recorder.CaptureByHandle(handleObject, "CaptureByHandle.png",
    "C:\\Documents\\ScreenRecorder")
||
    !ScreenRecorder::CaptureByHandle(handleString, "CaptureByHandle.tif",
    "C:\\Documents\\ScreenRecorder")
||
    !recorder.CaptureByHandle(handleString, "CaptureByHandle.bmp",
    "C:\\Documents\\ScreenRecorder")) {
  std::cerr << "ScreenRecorder::CaptureByHandle failed" << std::endl;
} // if (!CaptureByHandle(...))

system("pause");
return 0;
} // int main
This appendix appears in its original form, without editorial change.
Examples of using screen recording functions.

```cpp
#include <cstdio>
#include <iostream>
#include <sstream>
#include "ScreenRecorder.h"

// Link user32, gdi32, and gdiplus libraries
#pragma comment(lib, "user32")
#pragma comment(lib, "gdi32")
#pragma comment(lib, "gdiplus")

template <typename T>
std::string toString(T x) {
    // Converts x to a std::string
    std::stringstream sstr;
    sstr << x;
    return sstr.str();
}

int main(int argc, char* argv[]) {

    // **** ScreenRecorder Object Constructors ******************************************
    ScreenRecorder recorder1; // Default constructor
    ScreenRecorder recorder2(60.0, 10); // Specifying frame rate and recording time
    ScreenRecorder recorder3(60.0, 1.0); // Specifying time step and recording time

    // **** ScreenRecorder Getters and Setters ******************************************
    std::cout << "Default record time (s) = " << recorder1.GetRecordTime() << '\n';;
    std::cout << "Default frame rate (fps) = " << recorder1.GetFrameRate() << '\n';;
    std::cout << "Default time step (s)    = " << recorder1.GetTimeStep() << '\n';;

    // Output:
    // Default record time (s) = 60.0
    // Default frame rate (fps) = 1.0
    // Default time step (s)    = 1.0

    recorder1.SetFrameRate(2); // Also changes time step!
    std::cout << "New frame rate (fps)   = " << recorder1.GetFrameRate() << '\n';;
    std::cout << "New time step (s)      = " << recorder1.GetTimeStep() << '\n';;

    // Output:
    // New frame rate (fps) = 2.0
    // New time step (s)    = 0.5

    recorder1.SetRecordTime(120.0);
    recorder1.SetTimeStep(2.0); // Also changes frame rate!
    std::cout << "New record time (s)  = " << recorder1.GetRecordTime() << '\n';;
    std::cout << "New frame rate (fps) = " << recorder1.GetFrameRate() << '\n';;
    std::cout << "New time step (s)    = " << recorder1.GetTimeStep() << '\n';;
    std::cout << std::endl;
```

// Output:
// New record time (s)  = 120.0
// New frame rate (fps) = 0.5
// New time step (s)    = 2.0

// **** ScreenRecorder::RecordDesktop ***********************************************
if (!recorder1.RecordDesktop()) {
    std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
}

// RecordDesktop(fileNameBase)
if (!recorder2.RecordDesktop("RecordDesktop.bmp")) {
    std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
}

// RecordDesktop(fileNameBase, saveAsPath)
if (!recorder3.RecordDesktop("RecordDesktop.gif", "C:\Documents\ScreenRecorder")) {
    std::cerr << "ScreenRecorder::RecordDesktop failed" << '\n';
}

// **** ScreenRecorder::RecordByTitle ***********************************************
if (!recorder1.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]")) {
    std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
}

// RecordByTitle(windowTitle)
if (!recorder2.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]", "RecordByTitle.jpeg")) {
    std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
}

// RecordByTitle(windowTitle, fileNameBase)
if (!recorder3.RecordByTitle("Microsoft PowerPoint - [Sample.pptx]", "RecordByTitle.png", "C:\Documents\ScreenRecorder")) {
    std::cerr << "ScreenRecorder::RecordByTitle failed" << '\n';
}

// **** ScreenRecorder::RecordByHandle **********************************************
HWND handleObject = FindWindow(NULL, "Microsoft PowerPoint - [Sample.pptx]");

std::string handleString = toString(handleObject);

// RecordByHandle(windowHandle)
if (!recorder1.RecordByHandle(handleObject) || !recorder1.RecordByHandle(handleString)) {
    std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
}

// RecordByHandle(windowHandle, fileNameBase)
if (!recorder2.RecordByHandle(handleObject, "RecordByHandle.tif") || !recorder2.RecordByHandle(handleString, "RecordByHandle.bmp")) {
    std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
}
} // if (!recorder2.RecordByHandle(...))

// RecordByHandle(windowHandle, fileNameBase, saveAsPath)
if (!recorder3.RecordByHandle(handleObject, "RecordByHandle.gif",
"C:\Documents\ScreenRecorder")
!recorder3.RecordByHandle(handleString, "RecordByHandle.jpeg",
"C:\Documents\ScreenRecorder")
std::cerr << "ScreenRecorder::RecordByHandle failed" << '\n';
} // if (!recorder3.RecordByHandle(...))

system("pause");
return 0;
} // int main
Appendix C. ScreenRecorder.h

This appendix appears in its original form, without editorial change.
Interface for the ScreenRecorder class.

```cpp
// ************************************************************************************
// 12/01/2014 MKA                                            **************************
// ScreenRecorder.h: Interface for the ScreenRecorder class. ********************
// ************************************************************************************

#ifndef _SCREEN_RECORDER_H_
#define _SCREEN_RECORDER_H_

#include <sstream>
#include <string>
#include <windows.h>

#define TOL 1.0e-10
#define DEFAULT_MAXT 60.0
#define DEFAULT_DT 1.0
#define DEFAULT_FPS 1

class ScreenRecorder {
public:

    // **** Static Methods - Screen Capture *********************************************
    static bool CaptureDesktop(std::string saveAsFile = "ScreenCapture.png",
                                std::string saveAsPath = "");
    static bool CaptureByTitle(std::string title,
                                std::string saveAsFile = "ScreenCapture.png",
                                std::string saveAsPath = "");
    static bool CaptureByHandle(std::string handle,
                                 std::string saveAsFile = "ScreenCapture.png",
                                 std::string saveAsPath = "");
    static bool CaptureByHandle(HWND hWnd,
                                 std::string saveAsFile = "ScreenCapture.png",
                                 std::string saveAsPath = "");

    // **** Constructors ******************************************************
    ScreenRecorder();
    ScreenRecorder(double recordTime, int frameRate);
    ScreenRecorder(double recordTime, double timeStep);

    // **** Accessors ********************************************
    double GetRecordTime();
    double GetFrameRate();
    double GetTimeStep();

    // **** Mutators *****************************************************
    void SetRecordTime(double recordTime);
    void SetFrameRate(double frameRate);
    void SetTimeStep(double dT);

    // **** Non-Static Methods - Screen Record ****************************************
    bool RecordDesktop(std::string fileNameBase = "ScreenRecorder.png",
                        std::string saveAsPath = "");
    bool RecordByTitle(std::string title,
                        std::string fileNameBase = "ScreenRecorder.png",
                        std::string saveAsPath = "");

};
```
bool RecordByHandle(std::string handle,
    std::string fileNameBase = "ScreenRecorder.png",
    std::string saveAsPath = "");
bool RecordByHandle(HWND hWnd,
    std::string fileNameBase = "ScreenRecorder.png",
    std::string saveAsPath = "");

private:
    // **** Class Variables *********************************************************
    double fps; // framerate (frames per second)
    double dT; // timestep (sec)
    double maxT; // total time to record (sec)

    // **** Private Methods *********************************************************
    static std::string GetExeFileName();
    static std::string GetExePath();
    static wchar_t* GetWC(const char*c);
    static int GetEncoderClsid(const WCHAR* format, CLSID* pClsid);
    template<typename T> static std::string toLower(T str) {
        // Changes str to all lower case std::string
        unsigned int i;
        std::string newStr = static_cast<std::string>(str);
        for (i = 0; i < newStr.length(); ++i) {
            newStr[i] = tolower(str[i]);
        }
        return newStr;
    } // std::string toLower
    template<typename T>
    static std::string toString(T x) {
        // Converts x to a std::string
        std::stringstream sstr;
        sstr << x;
        return sstr.str();
    } // std::string toString
}; // class ScreenRecorder
#endif // ifndef SCREEN_RECORDER_H_
Appendix D. ScreenRecorder.cc

This appendix appears in its original form, without editorial change.
Implementation of the ScreenRecorder class.

```cpp
// 12/01/2014 MKA
// ScreenRecorder.cc: Implementation of the ScreenRecorder class.
// ************************************************************************************

#define _CRT_SECURE_NO_DEPRECATE
#include <ctime>
#include <direct.h>
#include <errno.h>
#include <windows.h>
#include <gdiplus.h>
#include <iostream>
#include <string>
#include "ScreenRecorder.h"

// ************************************************************************************
// **** Static Methods - Screen Capture **********************************************
// ************************************************************************************

bool ScreenRecorder::CaptureDesktop(std::string saveAsFile, std::string saveAsPath) {
    return CaptureByHandle(NULL, saveAsFile, saveAsPath);
} // bool ScreenRecorder::CaptureDesktop

bool ScreenRecorder::CaptureByTitle(std::string title, std::string saveAsFile, std::string saveAsPath) {
    HWND hWnd = FindWindow(NULL, TEXT(title.c_str()));
    if (!hWnd) {
        return false;
    } // if (!hWnd)
    return CaptureByHandle(hWnd, saveAsFile, saveAsPath);
} // bool ScreenRecorder::CaptureByTitle

bool ScreenRecorder::CaptureByHandle(std::string handle, std::string saveAsFile, std::string saveAsPath) {
    hWnd = (HWND) wcstoul(GetWC(handle.c_str()), NULL, 16);
    if (!hWnd) {
        return false;
    } // if (!hWnd)
    return CaptureByHandle(hWnd, saveAsFile, saveAsPath);
} // bool ScreenRecorder::CaptureByHandle

bool ScreenRecorder::CaptureByHandle(HWND hWnd, std::string saveAsFile, std::string saveAsPath) {
    Gdiplus::GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
    RECT rect = {0};
    int width = GetSystemMetrics(SM_CXSCREEN);
    int height = GetSystemMetrics(SM_CYSCREEN);
    if (saveAsPath.length() == 0) {
        saveAsPath = GetExePath();
    } // if (saveAsPath.length() == 0)
```
if (saveAsPath[saveAsPath.length()-1] != '\') {
    saveAsPath += '\';
} // if (saveAsPath[saveAsPath.length()-1] != '\')
wchar_t* fileName = GetWC((saveAsPath+saveAsFile).c_str());
std::string extension = saveAsFile.substr(saveAsFile.find_last_of('.') + 1,
    saveAsFile.length());

extension = toLower(extension);
if (extension == "jpg") {
    extension = "jpeg";
} else if (extension == "tif") {
    extension = "tiff";
} // adjust extension to account for common alternate format abbreviations
wchar_t* format = GetWC("image/" + extension).c_str();
HWND desktop = GetDesktopWindow();
HDC desktopHdc = GetDC(desktop);
HDC destHdc = CreateCompatibleDC(desktopHdc);
HDC iconHdc = CreateCompatibleDC(desktopHdc);
if (hWnd) {
    // If window handle was specified, then:
    // 1) bring window to front
    // 2) fix the window to stay on top and not change size or position
    // 3) update capture region size
    // Otherwise, capture the entire desktop
    if (IsIconic(hWnd)) {
        ShowWindow(hWnd, SW_SHOWNORMAL);
    } // if (IsIconic(hWnd))
    SetWindowPos(hWnd, HWND_TOPMOST, 0, 0, 0, 0,
        SWP_NOMOVE | SWP_NOSIZE | SWP_SHOWWINDOW);
    GetWindowRect(hWnd, &rect);
    // Double check that the window is valid, if not, then capture entire desktop
    if (rect.right >= rect.left) {
        width = rect.right - rect.left;
    } // if (rect.right >= rect.left)
    if (rect.bottom >= rect.top) {
        height = rect.bottom - rect.top;
    } // if (rect.bottom >= rect.top)
} // if (hWnd)

// Copy image to HBITMAP object
HBITMAP destBmp = CreateCompatibleBitmap(destHdc, width, height);
HBITMAP oldBmp = (HBITMAP)SelectObject(destHdc, destBmp);
BitBlt(destHdc, 0, 0, width, height,
    desktopHdc, rect.left, rect.top, SRCCOPY | CAPTUREBLT);
SelectObject(destHdc, oldBmp);

// Prepare to save in specified format (bmp, jpeg, gif, tiff, or png)
Gdiplus::Bitmap* newBmp = Gdiplus::Bitmap::FromHBITMAP(destBmp, NULL);
Gdiplus::Status status = Gdiplus::GenericError;
CLSID clsid;
if (newBmp && (GetEncoderClsid(format, &clsid) != -1)) {
    // Able to create Bitmap from HBITMAP and found valid clsid
    status = newBmp->Save(fileName, &clsid, NULL);
} // if (newBmp && (GetEncoderClsid(format, &clsid) != -1))

// Cleanup
if (hWnd) {
    SetWindowPos(hWnd, HWND_NOTOPMOST, rect.left, rect.top, width, height,
        SWP_SHOWWINDOW);
121 } // if (hWnd)
122 if (newBmp) {
123   delete newBmp;
124 } // if (newBmp)
125 Gdiplus::GdiplusShutdown(gdiplusToken);
126 ReleaseDC(desktop, desktopHdc);
127 DeleteObject(destBmp);
128 DeleteDC(destHdc);
129
130 return (status == Gdiplus::Ok);
131 } // bool ScreenRecorder::CaptureByHandle
132
133 // **********************************************************************************
134 // **** Constructors ****************************************************************
135 // **********************************************************************************
136 ScreenRecorder::ScreenRecorder() {
137   SetRecordTime(DEFAULT_MAXT);
138   SetTimeStep(DEFAULT_DT);
139 } // ScreenRecorder::ScreenRecorder
140
141 ScreenRecorder::ScreenRecorder(double recordTime, int frameRate) {
142   SetRecordTime(recordTime);
143   setFrameRate(static_cast<double>(frameRate));
144 } // ScreenRecorder::ScreenRecorder
145
146 ScreenRecorder::ScreenRecorder(double recordTime, double timeStep) {
147   SetRecordTime(recordTime);
148   SetTimeStep(timeStep);
149 } // ScreenRecorder::ScreenRecorder
150
151 // **********************************************************************************
152 // **** Accessors ********************************************************************
153 // **********************************************************************************
154 double ScreenRecorder::GetRecordTime() {
155   return maxT;
156 } // double ScreenRecorder::GetRecordTime
157
158 double ScreenRecorder::GetFrameRate() {
159   return fps;
160 } // double ScreenRecorder::GetFrameRate
161
162 double ScreenRecorder::GetTimeStep() {
163   return dT;
164 } // double ScreenRecorder::GetTimeStep
165
166 // **********************************************************************************
167 // **** Mutators *********************************************************************
168 // **********************************************************************************
169 void ScreenRecorder::SetRecordTime(double recordTime) {
170   if (recordTime < TOL) {
171     std::cerr << "ScreenRecorder: Record time must be >= 0.0" << "\n";
172     std::cerr << "Continuing with default record time (" << DEFAULT_MAXT << ")" << std::endl;
173     recordTime = DEFAULT_MAXT;
174   } // if (recordTime < 0.0)
175   maxT = recordTime;
176 } // void ScreenRecorder::SetRecordTime
void ScreenRecorder::SetFrameRate(double frameRate) {
    if (frameRate <= 0.0) {
        std::cerr << "ScreenRecorder: Frame rate must be > 0" << std::endl;
        frameRate = DEFAULT_FPS;
    }
    fps   = frameRate;
    dT    = 1.0 / fps;
}

void ScreenRecorder::SetTimeStep(double timeStep) {
    if (timeStep <= 0.0) {
        std::cerr << "ScreenRecorder: Time step must be > 0.0" << std::endl;
        timeStep = DEFAULT_DT;
    }
    dT    = timeStep;
    fps   = 1.0 / dT;
}

bool ScreenRecorder::RecordDesktop(std::string fileNameBase,
std::string saveAsPath) {
    return RecordByHandle(NULL, fileNameBase, saveAsPath);
}

bool ScreenRecorder::RecordByTitle(std::string title,
std::string fileNameBase,
std::string saveAsPath) {
    HWND hWnd = FindWindow(NULL, TEXT(title.c_str()));
    if (!hWnd) {
        return false;
    }
    return RecordByHandle(hWnd, fileNameBase, saveAsPath);
}

bool ScreenRecorder::RecordByHandle(std::string handle,
std::string fileNameBase,
std::string saveAsPath) {
    HWND hWnd = (HWND) wcstoul(GetWC(handle.c_str()), NULL, 16);
    if (!hWnd) {
        return false;
    }
    return RecordByHandle(hWnd, fileNameBase, saveAsPath);
}

bool ScreenRecorder::RecordByHandle(HWND hWnd,
std::string fileNameBase,
std::string saveAsPath) {
    clock_t   endTick;
    clock_t   t1;  
    clock_t   t2;
    errno_t   err;
    int       tickStep   = static_cast<int>(dT * CLOCKS_PER_SEC);
    int       frame      = 0;
    int       status;
bool wasSuccessful = true;
std::string saveAsFile;

if (saveAsPath.length() == 0) {
    saveAsPath = GetExePath();
} // if (saveAsPath.length() == 0)
if (saveAsPath[saveAsPath.length()-1] != '\\') {
    saveAsPath += '\\';
} // if (saveAsPath[saveAsPath.length()-1] != '\\')
saveAsPath += fileNameBase.substr(0, fileNameBase.find_last_of('.'));
saveAsFile = "\" + fileNameBase;

_set_errno(0);
status = _mkdir(saveAsPath.c_str());
_get_errno(&err);
wasSuccessful = ((status == 0) || (err == EEXIST));
saveAsPath += "\";

endTick = t1 + static_cast<int>(maxT * CLOCKS_PER_SEC);
t2 = t1;

while (wasSuccessful && (t2 < endTick)) {
    t2 = clock();
    if (((t2 - t1) > tickStep) {
        wasSuccessful = ScreenRecorder::CaptureByHandle(hWnd,
            toString(frame) + saveAsFile, saveAsPath);
        ++frame;
        t1 = t2;
    } // if ((t2 - t1) > tickStep)
} // while (wasSuccessful && (t2 < endTick))
return wasSuccessful;
} // bool ScreenRecorder::RecordByHandle

std::string ScreenRecorder::GetExeFileName() {
    char buffer[MAX_PATH];
    GetModuleFileName( NULL, buffer, MAX_PATH );
    return std::string(buffer);
} // std::string ScreenRecorder::GetExeFileName

std::string ScreenRecorder::GetExePath() {
    std::string f = GetExeFileName();
    return f.substr(0, f.find_last_of("\\" )+1);
} // std::string ScreenRecorder::GetExePath

wchar_t* ScreenRecorder::GetWC(const char *c) {
    const size_t cSize = strlen(c)+1;
    wchar_t* wc = new wchar_t[cSize];
    mbstowcs(wc, c, cSize);
    return wc;
} // wchar_t* ScreenRecorder::GetWC

int ScreenRecorder::GetEncoderClsid(const WCHAR* format, CLSID* pClsid) {
    // bmp, jpeg, gif, tiff, png
UINT num = 0; // Number of image encoders
UINT size = 0; // Size of the image encoder array in bytes
Gdiplus::ImageCodecInfo* pImageCodecInfo = NULL;
Gdiplus::GetImageEncodersSize(&num, &size);
if (size == 0) {
    return -1;  // Failure
} // if (size == 0)
pImageCodecInfo = (Gdiplus::ImageCodecInfo*)(malloc(size));
if (pImageCodecInfo == NULL) {
    return -1;  // Failure
} // if (pImageCodecInfo == NULL)
GetImageEncoders(num, size, pImageCodecInfo);
for (UINT j = 0; j < num; ++j) {
    if (wcscmp(pImageCodecInfo[j].MimeType, format) == 0) {
        *pClsid = pImageCodecInfo[j].Clsid;
        free(pImageCodecInfo);
        return j;  // Success
    } // if (wcscmp(pImageCodecInfo[j].MimeType, format) == 0)
} // for (UINT j = 0; j < num; ++j)
free(pImageCodecInfo);
return -1;  // Failure
} // int ScreenRecorder::GetEncoderClsid
INTENTIONALLY LEFT BLANK.
Appendix E. MATLAB Script for Creating AVI Movie from Screenshots

This appendix appears in its original form, without editorial change.
%% Sample script for creating an AVI movie from individual images

% Path to where images are saved
imagePath = 'C:\Documents\ScreenRecorder\';

% File base name
imageName = 'RecordByHandle.tif';

% Name of created video
movieName = 'SampleMovie.avi';

% Number of images to stitch together
numFrames = 50;

% Rate of playback for the video in frames per second
frameRate = 5;

% Create video writer object and set frame rate
writerObj = VideoWriter(strcat(imagePath, movieName));
writerObj.FrameRate = frameRate;

% Open video writer object for writing
open(writerObj);

% Write frames to video writer object
for k = 0 : numFrames-1
    fileName = strcat(imagePath, sprintf('%d%s', k, imageName));
    thisImage = imread(fileName);
    writeVideo(writerObj, thisImage);
end

% Close video writer object
close(writerObj);
INTENTIONALLY LEFT BLANK.