

Muffler Overview

Muffler is a MFC¹ Application generated in Visual Developer Studio 97 that serves as a front-end graphical user interface to a signal enhancement algorithm developed by Yariv Ephraim.

Acknowledgement

Implementation

The Muffler application has four integral parts. The first segment of Muffler is the main dialog, which allows a user to (1) specify an input / output file for enhancement operations, (2) start and stop the algorithm, and (3) play a specified input / output file. The second segment of Muffler is the options dialog, which allows a user to quickly and easily manipulate several signal-processing variables. The third segment of Muffler is an initialization (.ini) file, which allows an advanced user to (1) change, (2) load, and (3) save algorithm settings directly from the operating system. The final segment of the Muffler application is a Win32 Console Application created by a separate developer that houses the signal enhancement algorithm and processes an eight-bit sound file, based on the variables specified in the Muffler initialization file. A comprehensive tutorial of the Muffler application can be found in the Muffler User Guide.

Coding Analysis

Most of the source code in the Muffler application was created by AppWizard and ClassWizard, two interfaces provided with Visual Developer Studio which automatically generated source code for (1) activating dialog boxes, (2) linking objects such as message boxes, sliding bars, and menus to source code functions, and (3) synchronizing data with file objects. However, a substantial amount of source code was authored by the developer. Some examples of specialized source code include (1) a robust file input / output transfer system that filters out erroneous data, and (2) an abstract message routing system, which uses a string table and a constant header file to allow (a) quick algorithm / variable swapping for the possible implementation of a future enhanced algorithm, and (b) easy adaptation for an application version in a foreign language. A detailed study of the functions, properties, and plausible adaptation of Muffler can be found in the Muffler Programmer's Reference.

Algorithm Properties

¹ Microsoft Foundation Class

Although the Muffler application was designed to allow easy swapping of algorithms, it was based on the signal subspace speech enhancement algorithm created by Yariv Ephraim. Due to the technical nature of this algorithm, a synopsis / analysis of this algorithm will be given in the Muffler Algorithm Guide.

Revision History

The description of elementary projects leading up to Muffler

Appendixes

Appendix A: A brief introduction to digital signals

Legal Information

Acknowledgement

I would like to thank Dr. Howard Sabrin along with Atlantic Coast Technologies Inc., for allowing me to complete an internship in their Silver Springs locale. I would also like to thank Mr. Vien Nguyen for all his help in the course of the development of the Muffler MFC Application.

Muffler User Guide

This section provides information about the Muffler front-end GUI

- Muffler Dialog Diagrams
- Algorithm Variables
- Muffler User Tutorial
- Bug Reports
- Future Enhancements

Muffler Dialog Diagrams

This section of the user manual will list all significant parts of the two main dialogs in Muffler.

- Muffler Main Dialog (Startup Dialog)
- Enhancement Parameters Dialog (Variable Configuration Dialog)

Muffler Main Dialog

The following diagram is a guide to the Muffler Main Dialog

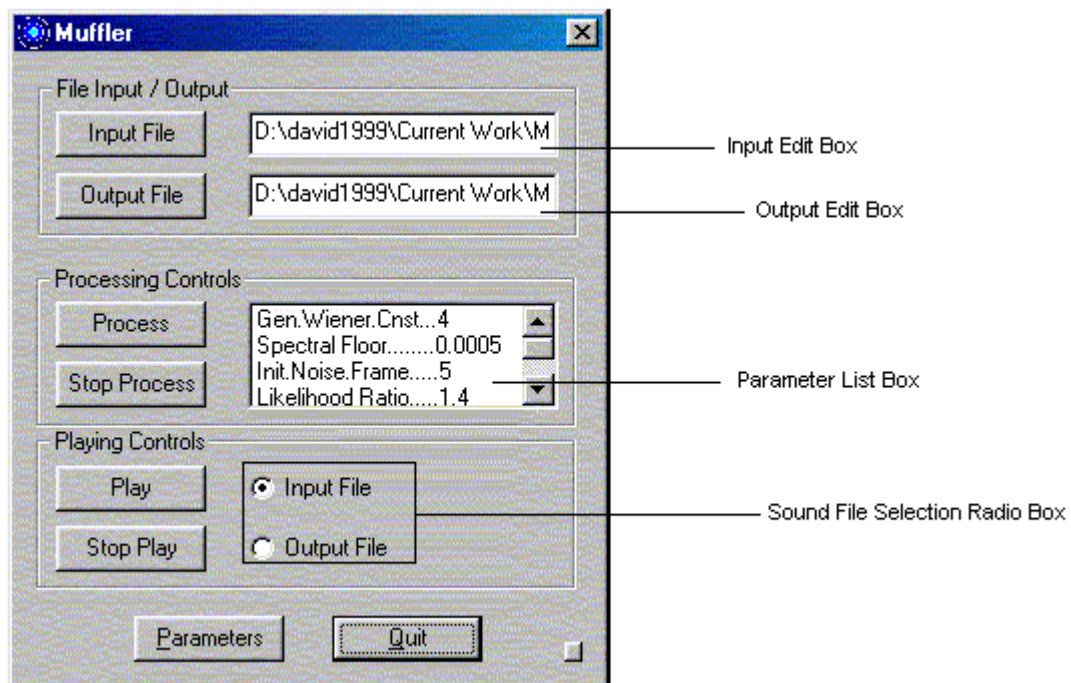


Figure 1

Enhancement Parameters Dialog

The following diagram is a guide to the Muffler Enhancement Dialog.

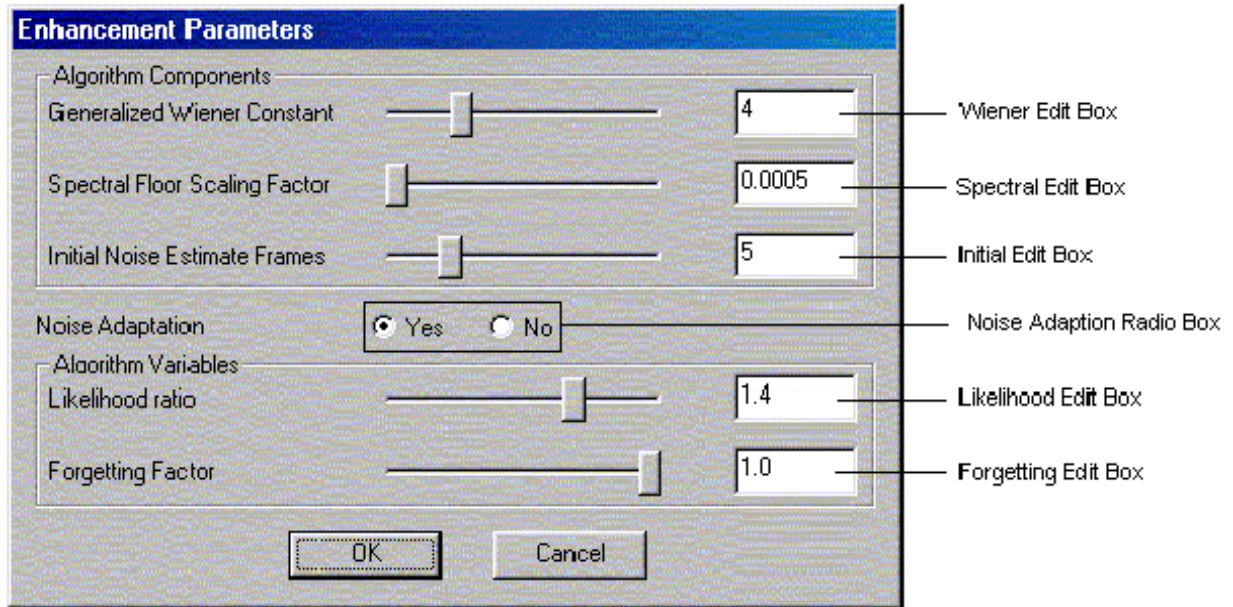


Figure 2

Muffler requires six main variables and two waveform audio files in order to operate correctly. In this section, the default values and ranges of each variable as well as the purpose of each waveform file will be given.

General Audio Note

Any audio file imported into Muffler must be a one-channel (monaural), eight-bit, pulse code modulation waveform audio file. Although Muffler will accept a stereo format waveform, it will process the waveform as a monaural sound. Therefore, if one channel in a stereo waveform is NOT disabled, the Muffler basal algorithm will concatenate both waveform channels and will probably output a highly distorted signal. Secondly, the parsing code in the Muffler basal algorithm first formats a waveform into its preferred format. If the a waveform file has been subject to low-level (assembly) changes, has been downloaded from some source with a 'packet' of missing data, or subjected to a lossy compression / decompression algorithm, the Muffler algorithm may prematurely overwrite the file. The creator of Muffler is not responsible for any loss of data. Secondly, Muffler is not capable of converting sound formats. For this primary reason, it is extremely risky to input any format other than an eight bit, monaural, PCM waveform audio sound file into Muffler.

Input File

The input waveform audio file is what Muffler uses as the initial sound file that it processes. In the processing of the sound file, Muffler does NOT alter the input audio file at all.

NOTE: If the sound file is extremely large (consists of more than 2^{16} bytes or longer than two minutes), the asynchronous playing function must load the whole sound file into the memory buffer or break up the sound and stream data to the buffer. In these two cases, once the sound starts, there will be a lengthy delay (approximately 30 seconds) before Muffler responds to a 'stop sound' command.

Output File

The output file is the enhanced version of the input file. An output file should be temporary, since Muffler will first clear the file in order to process the input file. In other words, any output file will be overwritten as soon as the 'Process File' button is clicked.

NOTE: Since Muffler does not support real-time processing, the input file and the output file must be different.

Generalized Wiener Constant

This variable is the first entry in the list box and is named 'Gen.Wiener Cnst'. The default value for this variable is 4. The minimum value for this variable is 2 and the maximum is 10. It has one decimal place.

Spectral Floor Scaling Factor

This variable is the second entry in the list box and is named 'Spectral Floor'. The default value for this variable is 0.0005. The minimum value for this variable is 0.0005 and the maximum is 1. It has four decimal places.

Initial Noise Estimate Frames

This variable is the third entry in the list box and is named 'Init.Noise Frame'. The default value for this variable is 4. The minimum value for this variable is 2 and the maximum is 10. It has one decimal place.

Noise Adaptation Radio Box

This variable determines if noise adaptation is enabled. If noise adaptation is enabled, two variables 'Likelihood ratio' and 'Forgetting Factor' are also enabled. If noise adaptation is disabled, then these two subsidiary variables are also disabled.

Likelihood Ratio

This variable is a subvariable that pertains to the noise adaptation radio box. If the noise adaptation radio button is set to 'Yes', the likelihood ratio is the fourth entry in the list box and is named 'Likelihood Ratio'. The default value for this variable is 1.4. The minimum value for this variable is 0 and the maximum is 2. It has one decimal place.

Forgetting Factor

This variable is a subvariable that pertains to the noise adaptation radio box. If the noise adaptation radio button is set to 'Yes', the likelihood ratio is the fifth entry in the list box and is named 'Forgetting Fctr'. The default value for this variable is 1.00. The minimum value for this variable is 0 and the maximum is 1. It has two decimal places.

Tutorial

As the purpose of Muffler is to enable a user to efficiently manipulate perform speech enhancement using Ephraim's signal subspace algorithm, this tutorial will demonstrate the normal usage of Muffler.

Processing A File

1. Click on the 'Input File' button or use the 'Input File Edit Box' to select an existing file.
2. Click on the 'Output File' button or use the 'Output File Edit Box' to select an existing temporary file / or a new sound.
3. After you've looked at the processing variable parameters, if you need to change them, double click in the list box or click on the button labeled 'Parameters'.
4. After re-checking the variables and the input / output files, click on the process button to initiate the algorithm.
5. After the processing step is completed, you may check the difference between both input and output sound files by selecting the appropriate choice in the sound file selection radio box and clicking the button labeled 'Play'.

Bug Report

Although Muffler is primarily robust, there are a few bugs in the application. Due to the stub interface that allows Muffler to receive run '*.exe' files as algorithms, Muffler cannot interface or communicate with its algorithm. This means that Muffler cannot (1) determine the progress of the algorithm and (2) halt the execution of the algorithm. However, this bug does not usually matter since the algorithm fully executes within a fraction of a second.

Future Enhancements

Although Muffler is fully capable as a speech enhancement application, it lacks some of the standard capabilities of standard released application. If Muffler is released in the future, it will be enabled with the following capabilities

- Embedded Spectrogram Display
- Embedded Waveform Frequency Display
- Background Thread Processing
- Real Time Operation
- Multiple Format Processing / Converting

Embedded Spectrogram Display

This capability will allow Muffler to display a spectrogram of a specified sound file.

Embedded Waveform Frequency Display

This capability will allow Muffler to display a waveform frequency of a specified sound file.

Background Thread Processing

This capability will allow Muffler to stop a stub application and not display the MS-DOS window at the same time.

Real Time Operation

This capability will allow Muffler receive sound input from an internal sound source (CD) or an external sound source (Microphone) and play or save to file an enhanced sound file simultaneously.

Multiple Format Processing / Converting

This capability will allow Muffler to process compressed sound files as well as sound in other formats (MPEG – 1,2,3; AVI; VOC;CD AUDIO)

Muffler Programmer's Reference

This section provides information about the programming component of the Muffler graphical user interface (GUI). Information is found under the following main headings:

- About Muffler
- Muffler Architecture
- Muffler Data File

About Muffler

The Muffler graphical user interface (GUI) is the end-user component of the Muffler application. The Muffler GUI provides low-latency error-checking, modular capacity, and direct operating system (OS) operability.

The Muffler GUI offers a 'no-learning curve' interface that allows an end-user to manipulate sound files and perform speech-enhancement. It also supports stub capabilities, which enable software developers to reroute algorithm-housing console applications into the graphical user interface without the modification of source code.

Muffler Architecture

This section introduces the components of Muffler and explains how the Muffler graphical user interface works with its stub application and operating system to process sound files. The following topics are discussed:

- Dialog Windows & Subsequent Properties
- Muffler Constants
- Muffler Classes
- Muffler Functions
- Muffler Stub File Integration

Dialog Windows & Subsequent Properties

The Muffler GUI contains three main dialogs: an 'about box' dialog, a startup dialog, and a variable manipulation dialog.

About Box

The 'About Box' is a standard dialog box that describes (1) what the application is, (2) who created it, and (3) copyright / disclaimer notices. In this specific graphical user interface (GUI), it is brought up in two ways (1) from the system menu (3) from the help menu in the startup dialog.

The Muffler ‘About Box’ is a rudimentary dialog box that contains one default ‘OK’ button, two basic static controls, which display the title of the application as well as a copyright notice, an ambient-sensitive icon, and an edit box which lists the disclaimer. The following illustration shows the one and only state of the about box.

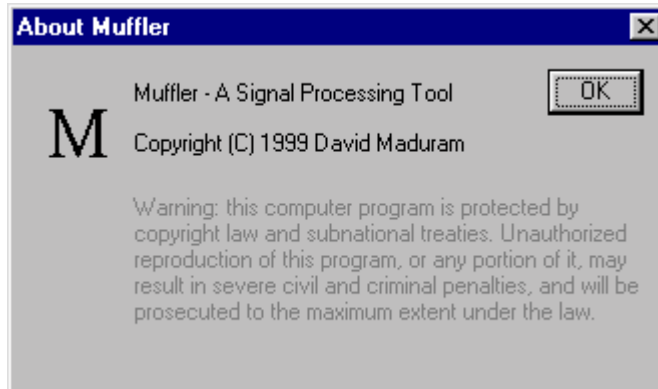


Figure 3

As one can see, the edit box is visible, disabled, and has multiline capabilities. The edit box cannot be selected and is not a tab stop. An edit box was chosen instead of a static text control for two main reasons. First of all, the copyright notice is 262 characters long (including whitespace characters) and thereby exceeds the 128 character limit for static controls. Second of all, an edit box was used in order to display a string from the string-table in the resource header. This would be needed if text animation was implemented or a foreign language version of Muffler was made.

The edit box’s control ID is ‘IDC_EDIT_BOX’ and is linked with two member variables, namely a CString type named ‘m_edit_box’ and a CEdit type named ‘m_edit_box’. Implementation of the text display process will be discussed later in the documentation.

This dialog box’s ID is ‘IDD_ABOUTBOX’. It has a ‘popup’ style, and a ‘dialog frame’ border. It also has a system menu and a title bar. Due to the simplistic nature of this dialog, there are no context-sensitive help IDs attached to this dialog box.

Startup Dialog

The startup dialog is the primary dialog of the application and is brought up when a user starts the application. Most often the only dialog box used by a beginning user, all high-level operations can be executed from this central dialog box. The Muffler startup dialog is divided by group boxes into three segments, namely “File Input / Output”, “Processing Controls”, and “Playing Controls”. An illustration of the one and only state of this dialog is displayed below.

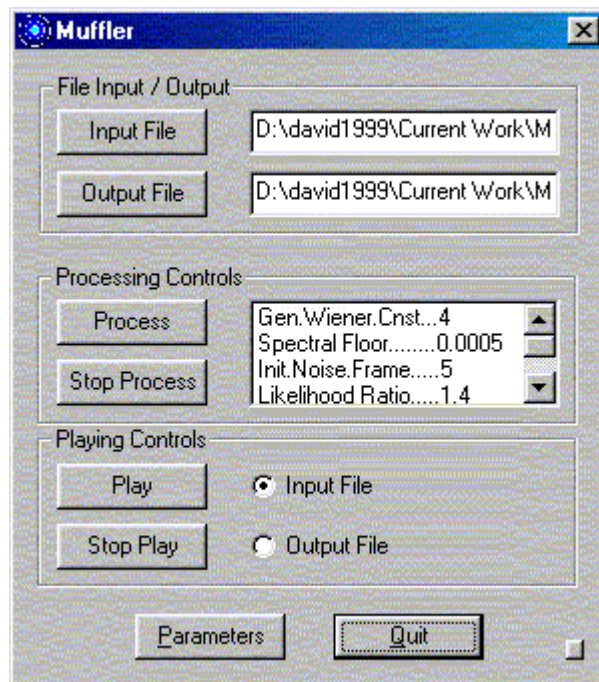


Figure 4

In the “File Input / Output” group box, there are four controls: two buttons and two edit boxes. The top two controls are linked to each other and work together to display the current input file. The ‘Input File’ ID is IDC_BUTTON_INPUT. The edit box’s ID is IDC_EDIT_INPUT. Although the ‘Input File’ button has no associated member variables, the Input File edit box has two. The edit box is associated with one CEdit variable called ‘m_edit_input_e’ and one CString variable called ‘m_edit_input’. The bottom two controls (e.g. ‘Output File’) are configured in the same way that its Input counterparts are. The edit box’s two member variables are ‘m_edit_output’ (CString) and ‘m_edit_output_e’ (CEdit).

In the “Processing Controls” group box, there are two controls. One is called ‘Process’ and the other is called ‘Stop Process’. Each button is linked with a message map

function called 'BN_CLICKED'. In this group box, there is also a list box. This control displays the present parameters. It has one member variable of type CListBox called 'm_listbox'.

Finally, in the "Playing Controls", there are five controls – four visible and one hidden. The two buttons 'Play' and 'Stop Play' are standard message-mapped buttons that each have a linked 'BN_CLICKED' message map function. One of the two radio button has a group property ('Input File'). It has an integer member variable entitled 'm_radio_input'. Both radio variables are linked to message map functions 'BN_CLICKED'. The hidden control is an edit box with a bool member variable. This control is primarily used by the radio buttons in order to communicate with the function classes.

Variable Manipulation Dialog

The variable manipulation dialog box is brought up in two ways (1) double-clicking on any item in the startup dialog's list box, or (2) clicking on the 'Parameters' button in the list box. The variable manipulation dialog, often referred to as the Parameters Dialog, is illustrated below.

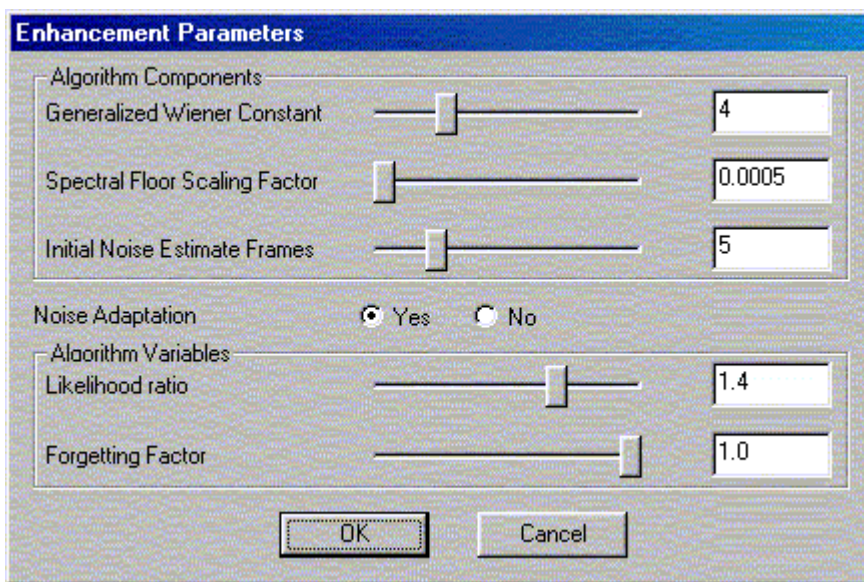


Figure 5

Each variable (e.g. Generalized Wiener Constant, Spectral Floor Scaling Factor, etc.) is paired with a slider and an edit box. The slider and the edit box are linked with each other with the windows messages "WM_ONHSCROLL" and edit box message "EN_CHANGE". Each slider is linked to a CSlider object with it's own prefix (e.g. 'spectral floor scaling factor' slider is linked with 'm_spectral_slider'). Finally, each edit box has two member variables, a CEdit and a CString. The noise adaptation

member variables are configured differently than the other radio buttons in the Startup dialog. Instead, each radio button has a group property with a CButton property. These allow the radio buttons to enable / disenable sliders and edit boxes.

Muffler Constants

Character Length Constants

There are two constants that pertain to character length. These constants are:

```
const int MAX_SLIDER_CHAR = 80;
const int MAX_FILEIO_CHAR = 942;
```

The first constant represents the maximum number of characters that can be entered into an edit box that corresponds to a slider (e.g. in Parameters dialog box). The second constant limits the maximum number of characters that can be passed to a file I/O buffer.

Radio Button Constants

There are two constants that pertain to default radio button settings. These constants are:

```
const double DEFAULT_PLAY_SETTING = 0
const double DEFAULT_NOISE_SETTING = 0
```

The first constant represents the default entry for the 'Play File' radio button group in the startup dialog. The second constant represents the default entry for the 'Noise Adaptation' radio button group in the parameters dialog.

Shift Constants

There are five constants that pertain to default multiple shift settings. These constants are:

```
const int WIENER_SHIFT = 1;
const int SPECTRAL_SHIFT = 10000;
const int INITIAL_SHIFT = 1;
const int LIKELIHOOD_SHIFT = 10;
const int FORGETTING_SHIFT = 100;
```

If one closely examines the parameters dialog, they might notice that the slider and edit box are linked. For instance, when a slider is moved, the numbers in the edit box move likewise. When numbers are typed into the edit box, the slider shifts accordingly. In order to support such an interface, the slider and the edit box must be able to communicate. However, there is one problem that prevents this linkage. All operations that deals with slider bars are integers. However, all the constants and variables, as

well as the units of the edit boxes, are doubles and must utilize decimal places. Therefore, in order to prevent data loss, each double is multiplied by a certain factor to form an whole integer. Now, data transactions can be executed easily, provided that there is a function that multiplies each variable with its successive multiple, as well as a function that can divide each variable with its successive multiple. The shift factor represents the multiplicative / divisible factor necessary for the data transaction.

Default Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double DEFAULT_NU = 4;  
const double DEFAULT_SPECTRAL = 0.0005  
const double DEFAULT_INITIAL = 5;  
const double DEFAULT_LHR = 1.4;  
const double DEFAULT_FORGETTING = .99;
```

At the initiation of the Muffler application, the slider / edit boxes are set to these values.

Generalized Wiener Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double WIENER_PG_SIZE = 2;  
const double WIENER_LN_SIZE = 1;  
const double WIENER_TIC_FRQ = 1;  
const double WIENER_MIN = 2;  
const double WIENER_MAX = 10;
```

At the initialization of the ‘Generalized Wiener’ slider bar, certain constants are set. PG_SIZE corresponds to the Page Size, or how much the slider moves when the buttons ‘Page Up’ and ‘Page Down’ are pressed. LN_SIZE corresponds to the Line size, or how much the slider moves when the left and right arrows are pressed. The tick frequency (TIC_FRQ) is an option that was implemented but not used. It, however, sets a tick mark at a certain interval throughout the slider bar. Finally, the range of the slider bar is set by ‘MIN’ and ‘MAX’, which are the minimum and the maximum.

Spectral Floor Scaling Factor Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double SPECTRAL_PG_SIZE = .2;  
const double SPECTRAL_LN_SIZE = .1;  
const double SPECTRAL_TIC_FRQ = .1;
```

```
const double SPECTRAL_MIN = .0005;  
const double SPECTRAL_MAX = 1;
```

At the initialization of the ‘Spectral Floor Scaling Factor’ slider bar, certain constants are set. PG_SIZE corresponds to the Page Size, or how much the slider moves when the buttons ‘Page Up’ and ‘Page Down’ are pressed. LN_SZE corresponds to the Line size, or how much the slider moves when the left and right arrows are pressed. The tick frequency (TIC_FRQ) is an option that was implemented but not used. It, however, sets a tick mark at a certain interval throughout the slider bar. Finally, the range of the slider bar is set by ‘MIN’ and ‘MAX’, which are the minimum and the maximum.

Initial Noise Estimate Frames Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double INITIAL_PG_SZE = 2;  
const double INITIAL_LN_SZE = 1;  
const double INITIAL_TIC_FRQ = 2;  
const double INITIAL_MIN = 1;  
const double INITIAL_MAX = 20;
```

At the initialization of the ‘Initial Noise Estimate Frames’ slider bar, certain constants are set. PG_SIZE corresponds to the Page Size, or how much the slider moves when the buttons ‘Page Up’ and ‘Page Down’ are pressed. LN_SZE corresponds to the Line size, or how much the slider moves when the left and right arrows are pressed. The tick frequency (TIC_FRQ) is an option that was implemented but not used. It, however, sets a tick mark at a certain interval throughout the slider bar. Finally, the range of the slider bar is set by ‘MIN’ and ‘MAX’, which are the minimum and the maximum.

Likelihood Ratio Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double LIKELIHOOD_PG_SZE = .2  
const double LIKELIHOOD_LN_SZE = .1  
const double LIKELIHOOD_TIC_FRQ = .1  
const double LIKELIHOOD_MIN = 0;  
const double LIKELIHOOD_MAX = 2;
```

At the initialization of the ‘Likelihood Ratio’ slider bar, certain constants are set. PG_SIZE corresponds to the Page Size, or how much the slider moves when the buttons ‘Page Up’ and ‘Page Down’ are pressed. LN_SZE corresponds to the Line size, or how much the slider moves when the left and right arrows are pressed. The tick frequency (TIC_FRQ) is an option that was implemented but not used. It, however, sets a tick mark at a certain interval throughout the slider bar. Finally, the range of the slider bar is set by ‘MIN’ and ‘MAX’, which are the minimum and the maximum.

Forgetting Factor Constants

There are five constants that pertain to original edit box / slider settings. These constants are:

```
const double FORGETTING_PG_SIZE = .2;
const double FORGETTING_LN_SIZE = .1;
const double FORGETTING_TIC_FRQ = .1;
const double FORGETTING_MIN = 0;
const double FORGETTING_MAX= 1;
```

At the initialization of the ‘Forgetting Factor’ slider bar, certain constants are set. PG_SIZE corresponds to the Page Size, or how much the slider moves when the buttons ‘Page Up’ and ‘Page Down’ are pressed. LN_SIZE corresponds to the Line size, or how much the slider moves when the left and right arrows are pressed. The tick frequency (TIC_FRQ) is an option that was implemented but not used. It, however, sets a tick mark at a certain interval throughout the slider bar. Finally, the range of the slider bar is set by ‘MIN’ and ‘MAX’, which are the minimum and the maximum.

Muffler Classes

The Muffler GUI contains five classes. The first four classes “CAboutDlg”, “CDialogProperties”, “CMufflerApp”, and “CMufflerDlg” are all AppWizard created. The last class, “Data” a is user created class.

CAboutDlg manages the ‘About Muffler’ dialog box and loads the copyright law string, CDialogProperties manages the ‘Parameters’ dialog box and all of its sliders / edit boxes, CMufflerApp and CMufflerDlg both manage the startup dialog, and Data manages data serialization between Muffler and the stub file’s batch file.

Muffler Functions

The Muffler GUI contains approximately twenty functions. Most of these functions have been modified in some way or another to make Muffler work. Shown below is a table of the functions that were predominantly created by myself along with a brief description of the purpose of the function.

Data Transmission	
CDialogProperties::onOK	Sends current properties to file
CMufflerDlg::ConvertData()	Load data into Data() class
CMufflerDlg::UpdateData()	Write data into file from startup dialog
CMufflerDlg::UpdateList()	Write data into list box straight from file

Sound Verification / Playing	
CMufflerDlg::OnButtonPlay()	Play a sound (PlaySound ~ ASYNCH)
CMufflerDlg::OnButtonStopPlay()	Play a null sound to stop present sound
CMufflerDlg::VerifySound()	Check if sound can be played (Sound Processing subroutine)

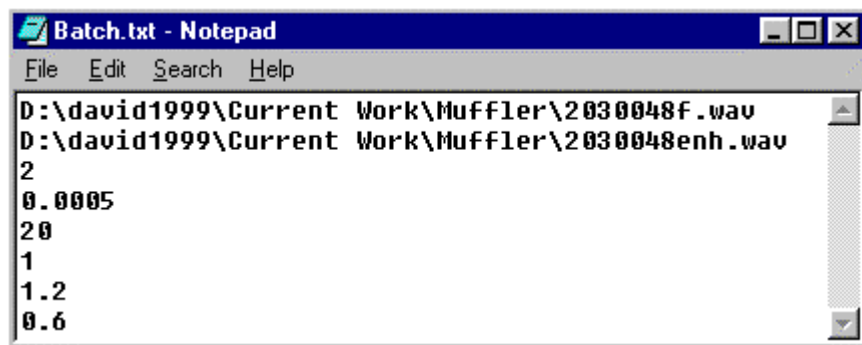
Muffler Stub File Integration

The Muffler GUI used a single command to run the Win32 Console Application. The command 'system'² enables a developer to run a program just as if one enters a command at the MS-DOS command prompt. This command enabled me to process a sound when a user clicked the 'Process' button.

² Special thanks to Vien Nguyen for this function

Muffler Data File

The Muffler GUI communicates with the back-end algorithm executable using a batch file. The batch file is illustrated below.



```
Batch.txt - Notepad
File Edit Search Help
D:\david1999\Current Work\Muffler\2030048f.wav
D:\david1999\Current Work\Muffler\2030048enh.wav
2
0.0005
20
1
1.2
0.6
```

Figure 6

Line 1: Input File

Line 2: Output File

Line 3: Generalized Wiener Constant

Line 4: Spectral Floor Scaling Factor

Line 5: Initial Noise Estimate Frames

Line 6: Noise Adaptation Boolean Representation

Line 7: Likelihood Ratio

Line 8: Forgetting Factor

Muffler Algorithm Guide

This section provides information about the signal subspace algorithm (SSA) that the Muffler application is based on. Information is found under the following main headings:

- Introduction to Speech Enhancement
- Algorithms Used in Speech Enhancement
- Speech Enhancement via SSA
- References

Introduction to Speech Enhancement

The goal of a speech is to convey a certain message from one person to another. This transmission of data can be described as the communication of raw information. Alternatively, this data communication can also be characterized as the acoustic waveform that carries this information. This acoustic waveform is generally known as a signal.

In short-range communications (e.g. unaided two-person configuration), transmitting information via speech is generally foolproof (the message information is successfully transferred from the speaker to the listener). However, in long-range communications that involve a third party system to transfer information, transmitting information via speech can be less reliable. This is usually caused by *noise*. Noise can be described as the part of the acoustic waveform that (1) is unintentionally created and (2) does not pertain to the message information.

Noise can be created by two main sources. The first source of noise is ambient noise. An example of ambient noise is the background noise (vehicles, unrelated human conversations, etc.) in an urban setting. The second source of noise is channel noise. Channel noise is divided into two categories based on the medium through which the sound signal is transmitted. In a medium like water, air, or a vacuum, the channel noise is classified as softwire noise. Some examples of softwire noise are reverberations, ‘rippling’ noises, and sound distortions caused by high pressure. If the sound signal is transmitted through a medium like coaxial cables, telephone lines, or satellite cables, the channel noise is classified as hardwire noise. Natural electric disturbances (lightning), satellite transponders, and static are all examples of hardwire noise. As one can see, channel noise is wholly dependent on its medium of signal transfer. The third source of noise is frequency dependent noise.

The goal of sound enhancement is to remove noise from a particular signal. Speech enhancement is a specific subsection of sound enhancement. Speech enhancement has a three-fold nature: (1) to remove extraneous noise, (2) improve the quality of the signal, and (3) improve the intelligibility of the signal. Quality can be defined as the pleasantness of the sound signal. Quality is important in speech enhancement because

this factor decreases listener fatigue and aids intelligibility. Intelligibility can be defined as the percentage of raw information successfully transmitted from the speaker to the listener.

Algorithms used in Speech Enhancement

There are five substantial algorithms currently used in speech enhancement. This section will give a brief description of each algorithm.

Lowpass / Highpass

A lowpass filter is a linear time-invariant (LTI) method that blocks high-pitched frequencies. A highpass filter is a LTI method that blocks low-pitched frequencies.

Bandpass / Bandstop

A bandpass filter is also a LTI method that blocks all frequencies outside a certain range. A bandstop filter is a LTI method that blocks all frequencies inside a certain range.

Spectral Subtraction

A spectral subtraction method works by taking the power spectral density of a signal when the speaker is talking (clean signal + noise) and when the speaker is not talking (clean signal) and subtracting the two.

Speech Enhancement via SSA

The signal subspace algorithm is an altered version of the spectral subtraction method that has proved to be extremely effective in speech. Below is an illustration of a waveform shown before enhancement and after enhancement.

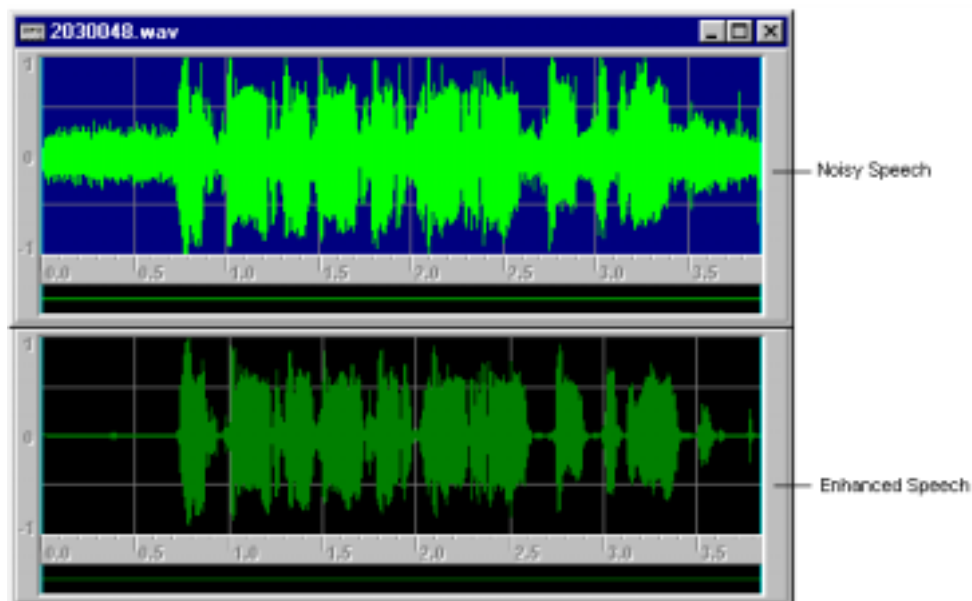


Figure 7

Muffler Revision History

Version 1.0	
Internal Name	Muffler
Language	C++
Developer	David Maduram
Comments	First Release
Purpose	Stub Application (Inclusion of a separate executable file) Menu-Driven System
Date Completed	October 25, 1999

Version 0.9	
Internal Name	Helper
Language	C++
Developer	David Maduram
Comments	None
Purpose	Right-Click Help (WM_CONTEXTMENU) Indexing / Keyword Help (WinHelp) Normal / F1 Help (WM_HELPINFO)
Date Completed	November 11, 1999

Version 0.8	
Internal Name	Project1
Language	C++
Developer	David Maduram
Comments	First MFC application with sound interface ability
Purpose	Familiarization with File Input / Output (CFileDialog) Sound input / output (PlaySound() and "winmm.lib") Multithreading and thread synchronization (mutex)
Date Completed	November 19, 1999

Version 0.7	
Internal Name	Propsheet
Language	C++
Developer	David Maduram
Comments	First multiple-dialog MFC application
Purpose	Familiarization with multiple classes in Visual Studio Dialog properties (Child and Thin) SDI (Single Document Interface) operation
Date Completed	November 24, 1999

Version 0.6	
Internal Name	Dialog1
Language	C++
Developer	David Maduram
Comments	First working MFC application
Purpose	Creating and manipulating grouped radio boxes Adding items to list boxes Alert messages with <code>AfxMessageBox()</code> and <code>AfxMessage()</code> Message Maps (ClassWizard operational development)
Date Completed	November 26, 1999

Version 0.5	
Internal Name	Speech Enhancement
Language	C++
Developer	David Maduram
Comments	First version created by primary developer
Purpose	Familiarization with Visual Developer Studio 97 and AppWizard
Date Completed	November 27, 1999

Version 0.4	
Internal Name	SpEnhancement
Language	MATLAB
Developer	Yariv Ephraim
Comments	First original version
Purpose	Model for subsequent versions
Date Completed	Not Applicable

Appendix A

Mathematically, a signal can be represented as acoustic pressure as a function of time. This is illustrated in the following diagram.

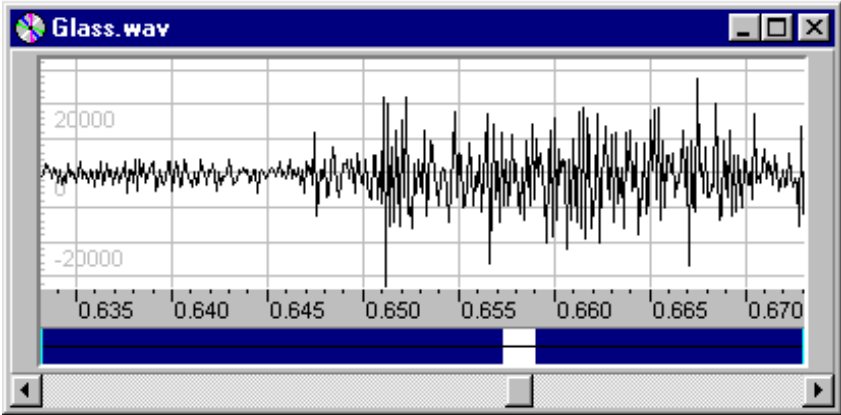


Figure 8

Usually, when a sample is digitally synthesized, the continuous waveform is broken up into segments called samples. Each sample holds one piece of data, which is the amplitude of the signal at a specific time. As one can clearly see, a digitally synthesized signal is the result of a discrete concatenation of these samples.

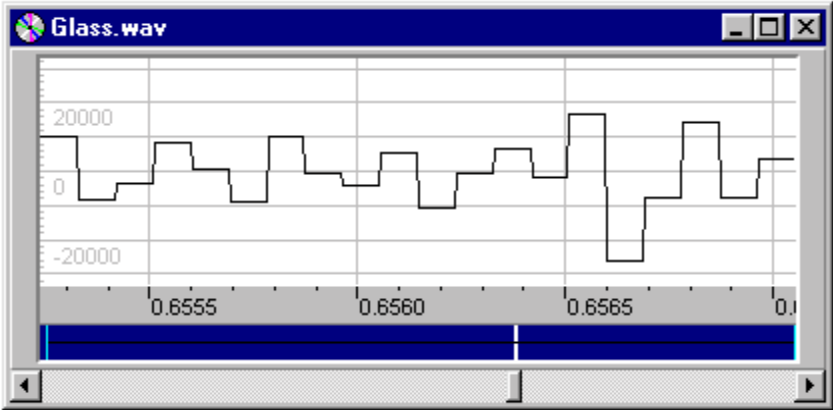


Figure 9

A digitally synthesized signal has several characteristics: sampling rate, number of bits, and number of channels.

The sampling rate is the number of times that the amplitude level is recorded. This can be interpreted as the number of samples generated per second. When the sampling rate increases, the signal clarity increases, which means that the amount of data stored per second also increases. Most digital recording schemes use several standard sampling rates in order allow a tradeoff between data storage and signal detail. These schemes will be discussed later in this paper.

The number of bits determines how accurately the amplitude of a sample is recorded. A piano, for instance, uses a one-bit coding schema: it can only play sounds differentiated by approximately 45 Hertz. On the other hand a violin has the ability (due to its string interface) to generate sounds³ that are differentiated with a frequency below 45 hertz. Likewise, a signal digitization configuration that uses a sixteen-bit decoding schema (2^{16} different levels of amplitude) will naturally have more precision than an eight-bit decoding schema (2^8 different levels of amplitude). Of course, the more bit precision⁴ is used, the memory needed *per sample* increases also.

The number of channels determines how many sound tracks are supported by the signal. For instance, a signal with one channel (known as mono / monoaural) contains only one sound track. A signal with two channels (known as stereo / stereo audio) has information for two sound tracks. Even though multiple channels can provide three-dimensional sound and is imperative in modeling life-like sound, there is a high memory / data cost in its implementation. For instance, a two channel signal requires twice the storage / memory of a one channel signal.

³ Although the classical musical system imposes a 45 Hertz limit (difference between an 'A' and 'A sharp') on most works, the violin sometimes generates sounds between this set frequencies through the use of glissando.

⁴ Since the human ear usually cannot differentiate between sound separated by less than .00001 Hertz, a sixteen bit is sufficient for optimal signal processing / playback.

Legal Information

Information in this document is subject to change without notice. The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of either Atlantic Coast Technologies or David Maduram.

Portions of this document specify and accompany software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of final documentation or software. David Maduram assumes no responsibility for any damages that might occur directly or indirectly from these inaccuracies.

Atlantic Coast Technologies may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Atlantic Coast Technologies, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 1999 David Maduram. All rights reserved.

Microsoft, MS-DOS, MS, Windows, ClassWizard, AppWizard, Visual Developer Studio 97, Help Workshop, ActiveX, DirectX, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

SpEnhancement © Copyright 1995 Yariv Ephraim.

CoderGUI © Copyright 1999 Atlantic Coast Technologies.

ACTI is a registered trademark of Atlantic Coast Technologies.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.