

Notey

A real-time music notation detection software



Hila Shmuel

notey.recorder@gmail.com

September 2011

<https://sites.google.com/site/noteysoftware/>

Table of Contents

Project Participants.....	4
Project Description.....	5
Goal.....	5
Features.....	5
Research & Algorithms.....	6
Program Main Algorithm.....	6
Research Bibliography.....	6
Sampling and Detecting a Musical Note.....	7
Musical Knowledge.....	12
Notes Detection Over Time.....	15
ABC Music Notation.....	16
Technical Problems & Solutions.....	17
Recording While Displaying.....	17
Displaying & Updating Music Sheet in Real-Time.....	18
Recorder Hero feature - implementation.....	19
Project QA.....	20
Application User Guide.....	22
Installation.....	22
Installation Requirements.....	24
Setting Configuration.....	25
Setting GhostScript.....	26
Setting ABCM2PS.....	26
Recording.....	27
Start Recording.....	28
Stop Recording.....	28
New Recording.....	28
Real-Time Recorded Notes Music Sheet.....	29
Real-Time WaveForm.....	30
PDF Music Sheet.....	31
Note Data and Fingering.....	32
Playing.....	33

Saving a Recording.....	34
Open Music Notes File (RecorderHero).....	35
open an ABC file.....	35
Start and Pause the RecorderHero.....	35
Use the 'Start Recording' and 'Stop Recording' buttons. Select a microphone and start playing (see Start Recording). In order to start over, re-open the file.....	35
Speed Up/Down the RecorderHero.....	35
Using the RecorderHero.....	36
Interactive Music Sheet.....	37
Application Developer Guide.....	38
Compilation Instruction & Requirements.....	38
Program Main Flow.....	39
Recording thread.....	39
RecorderHero Thread.....	48

Project Participants

Project Coordinator

Name: Hananel Hazan

Email: hhazan01 AT cs.haifa.ac.il

Institution: Department of Computer Science at the University of Haifa

Project Advisor

Name: Avner Ben-Shimol

Project Developer

Name: Hila Shmuel

Email: notey.recorder AT gmail.com

Institution: Department of Computer Science at the University of Haifa

Project Description

Goal

"**Notey** is a real-time notes detector.

It allows you to sit next to the computer microphone, and play in the Recorder (music instrument), and the notes will appear on the music sheet, in real time!

With **Notey** you can also open a music notes file (in ABC format), and start playing - and like in GuitarHero (a console game), you will be able to see the notes you should play and if you had played them correctly, again, in real-time!"

While there are many musical instruments tuners software online, the simple need of every musician – to write down his music – is not being answered by them.

Notey comes to answer that need. For musicians and beginners, **Notey** makes its simple - one can play in his Recorder and the notes will appear on screen. The person who uses the software don't even have to know how to play in the instrument or how to write note – he can learn it by using the software.

Features

- **A real time note detectors** - print the notes you play on music sheet, in real-time
- **Notes tracker (like in GuitarHero, but for every music sheet)** - opening an ABC (music notation file), and start to play according to the notes running on screen. like in GuitarHero, you will be able to see the notes you should play and if you had played them correctly, again, in real-time!
- plays the notes recorded as Wave, or Beeps (via computer speaker).
- shows the sound wave and frequency of the notes been played.
- save the played notes, in the formats: WAV, ABC, PDF, PostScript.
- display an interactive music sheet – for beginners.

Research & Algorithms

Program Main Algorithm

The main goal of the project is to detect notes played on real-time.

Starting to design such a project, we can think of the following flow:

1. Record a waveform sound sample, in real time.
2. perform an analysis to extract the current played note from the sample.
3. Given that note, it's not trivial that this note is the note being currently played – it may be a random noise. We must perform an analysis of **notes detection over time**.
4. Display the new note on music sheet.

While (1) and (4) are technical issues and to be discussed in the next chapter ("Technical Problems & Solutions"), steps (2) and (3) are pure computer science and DSP (digital signal processing) field issues.

Research Bibliography

For a better understating of the sound analysis area, I've read chapters 8-13, 22(about FFT and sound processing) from the GREAT book:

- "The Scientist and Engineer's Guide to Digital Signal Processing" By Steven W. Smith, Ph.D.

For the music knowledge, I've mostly used musicians with record of years, and Wikipedia.

For the ABC notation knowledge, I've used the ABC site.

<http://abcnotation.com/>

Sampling and Detecting a Musical Note

The following is based on the book: "The Scientist and Engineer's Guide to Digital Signal Processing" By Steven W. Smith, Ph.D.

In-order to accomplish the project goal, first of all, we must solve this problem:

Problem: Given an array of samples of a sound waveform, extract the main frequency.

Solution: FFT (Fast Fourier transform).





Fast Fourier transform is a wide-known mathematical algorithm for computing the **discrete Fourier transform** (DFT) and its inverse. A **DFT** decomposes a sequence of values into components of different frequencies.

Why DFT?

A signal can be either continuous or discrete, and it can be either periodic or aperiodic. The combination of these two features generates the four categories:

Aperiodic-Continuous, Periodic-Continuous, Aperiodic-Discrete, Periodic-Discrete.

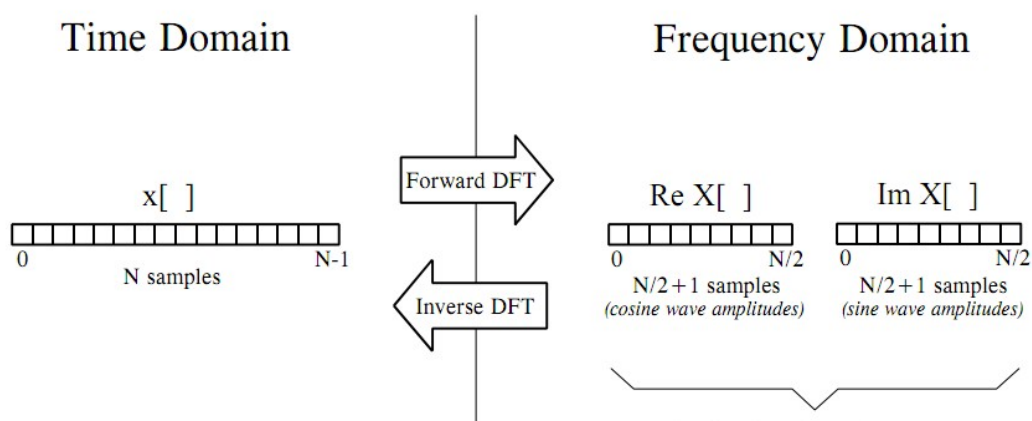
For our goal, the waveform sound signal is **Periodic-Discrete** - These are discrete signals that repeat themselves in a periodic fashion from negative to positive infinity. This class of Fourier Transform is sometimes called the Discrete Fourier Series, but is most often called the **Discrete Fourier Transform**.

Type of Transform	Example Signal
Fourier Transform <i>signals that are continuous and aperiodic</i>	
Fourier Series <i>signals that are continuous and periodic</i>	
Discrete Time Fourier Transform <i>signals that are discrete and aperiodic</i>	
Discrete Fourier Transform <i>signals that are discrete and periodic</i>	

In fact, this is the only type of transform that can be processed by computers – since in computer we can only represent a limited set of data (hence **Discrete**) and in digital signal processing, the sound wave are **Periodic**.

How Does it Work? (the real DFT)

there are two algorithms versions of the DFT – real DFT and complex DFT. For our purpose we will use the real DFT, which is much simpler.



Input - Time Domain: an array of N point input signal, contains the signal to be decomposed. The name 'Time Domain' is because usually, the input will be samples taken at regular intervals of time. Refer as $\mathbf{X}[]$.

Output – Frequency Domain: two array of $N/2+1$ points:

1. Real part of $X[]$ - **ReX[]** - contains the amplitudes of the component cosine waves.
2. Imaginary part of $X[]$ - **ImX[]** - contains the amplitudes of the component sine waves.

N is chosen to be a power of two: ...1024, 2048,... because of the FFT algorithm (described later).

The DFT basis functions are generated from the equations:

$$c_k[i] = \cos(2\pi ki/N)$$

$$s_k[i] = \sin(2\pi ki/N)$$

c_k[] is the cosine wave for the amplitude held in **ReX[k]**, and

s_k[] is the cosine wave for the amplitude held in **ImX[k]**.

In simple words, for example, $c_5[]$ is the cosine wave that makes 5 complete cycles in N points, so **ReX[5]** will hold its amplitude. So, **ReX[k]** will hold the amplitudes of the **cosine** waves that makes from 0 to $N/2$ complete cycles in N points, and **ImX[k]** will hold the amplitudes of the **sine** waves that makes from 0 to $N/2$ complete cycles in N points.

Calculating the DFT by Correlation

This is the formula to calculate the frequency domain from the time domain:

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

$$ImX[k] = - \sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

Each sample in the frequency domain is found by multiplying the time domain signal by the sine or cosine wave being looked for, and adding the resulting points.

This is work due to the concept of correlation – in order to detect a known waveform contained in another signal, multiply the two signals and add all the points in the resulting signal. The single number that results from this procedure is a measure of how similar the two signals.

Why FFT? (Fast Fourier transform)

In this project we use the FFT implementation of the DFT algorithm.

Computing the DFT by using the math equations above, is a slow operation – $O(N^2)$ arithmetical operations.

The FFT can compute the same result in only $O(N \log N)$ operations.

And indeed, while testing the project, the two algorithms were tested – and for big data-sets (2048 or 4096 samples in array) the result was

From sine & cosine to Hertz

Given the DFT output, we now have two arrays – $ReX[]$ and $ImX[]$, in what called the **rectangular form**. But how can we get the frequencies in Hertz? For that purpose we use the **polar form** of the DFT output. In this form, $ReX[]$ and $ImX[]$ are replaced with two other arrays: Magnitude of $X[]$ and Phase of $X[]$, or: **MagX[]** and **PhaseX[]**.

$$MagX[k] = (ReX[k]^2 + ImX[k]^2)^{1/2}$$

$$PhaseX[k] = \arctan\left(\frac{ImX[k]}{ReX[k]}\right)$$

$$ReX[k] = MagX[k] \cos(PhaseX[k])$$

$$ImX[k] = MagX[k] \sin(PhaseX[k])$$

Those two forms (polar and rectangular) are equal. So, why to use the polar form?

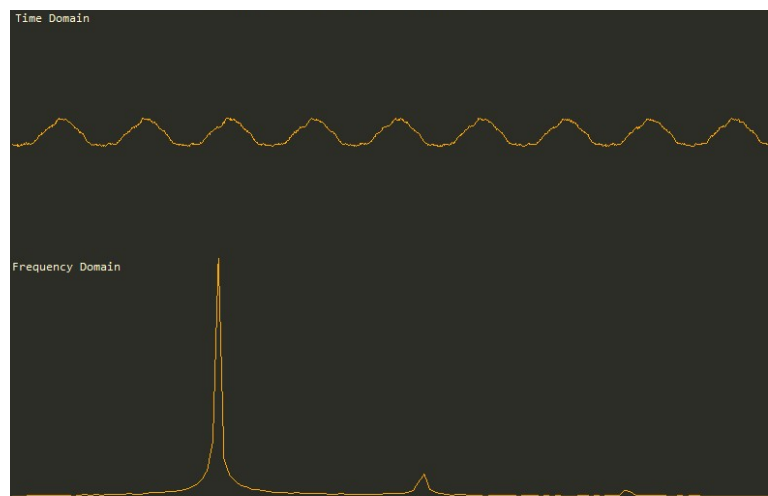
With rectangular notation, the DFT decomposes an N point signal into $N/2 + 1$ cosine waves and sine waves, each with a specified amplitude.

With polar notation, the DFT decomposes an N point signal into cosine $N/2 + 1$ waves, each with a specified amplitude (called the magnitude) and phase shift.

For our purpose (DSP), we ignore the phase shift element, and use only the magnitude. This gives as a smooth graph, that from it we can calculate the frequencies in Hertz.

The calculation itself is to be described next.

This is a real time analysis of FFT in the program



Musical Knowledge

Sound is vibration. Microphone can pick up these vibrations and translate them into electrical currents, that can be represented by a waveform that shows vibrations over time (by PCM). The most natural form of vibration is represented by the sine wave.

The sine wave has two parameters - **amplitude** (loudness) and **frequency** (pitch).

In the next text, a few basic musical concepts will be covered.

Pulse Code Modulation

PCM is a method of storing the waveform on computer. It has two parameters:

1. **sample rate**, or how many times per second you measure the waveform amplitude.
2. **sample size**, or the number of bits you use to store the amplitude level.

With pulse code modulation, a waveform is sampled at a constant periodic rate, usually some tens of thousands of times per second. For each sample, the amplitude of the waveform is measured. The hardware that does the job of converting an amplitude into a number is an analog-to-digital converter (ADC).

Sampling Rate (Nyquist frequency)

The sampling rate determines the maximum frequency of sound that can be recorded . According to the "Nyquist–Shannon sampling theorem", The sampling rate must be twice the highest frequency of sampled sound.

So, if the recorder instrument highest frequency is 3000Hz, the sampling rate must be at list 6000 samples per second. However, because low-pass filters have a roll-off effect, the sampling rate should be about 10 percent higher than that.

FFT frequency domain result in Hertz

Using the FFT algorithm, we got back an array of frequencies. But what is their range in Hertz?

For **N** input samples, sampled at **SampleRate** (samples per second), the result is [0,..., N/2] frequencies back , that should be normalized by the following formula:

$$\text{FrequencyInHertz} = \text{SampleRate} / N * i \text{ for } i \text{ in } [0, ..., N/2]$$

Note Frequency

Given a the main frequency of the waveform in Hertz, we must recognize if it's a musical note's frequency or not.

This is done by generating a table of notes and their corresponding frequencies, and checking whether the given frequency matches a note in the table.

Using this method, we use the MIDI music notation specification – which assign a number [0, 127] to each note. **n** is the MIDI index:

$$f = 440 \times 2^{(n - 69) / 12} \quad \text{for } n \text{ in } [0, 127]$$









With set of notes in each octave : C, C#, D, D#, E, F, F#, G, G#, A, A#, B

This produces a table:

Frequency (Hz)	MIDI (n)	Note	Octave
8.175799	0	C	-1
8.661957	1	C#	-1
.			
.			
.			
12543.85	127	G	9

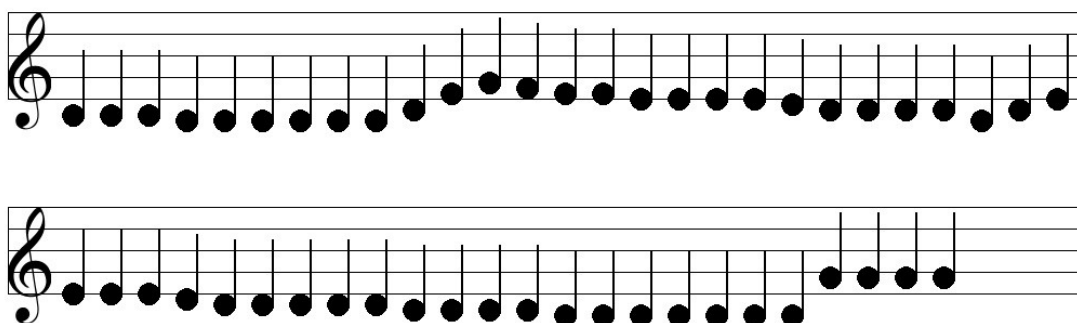
Note Lengths

Each note in the music sheet has its own duration. This duration is not an absolute one (it does not define how many seconds it should last), but is relative to the speed of the beats). The following table gives the duration which is the number of seconds that one note would last relative to a Whole note of 64 time units:

USA Note Name	Note	Duration
Double whole note		128
Whole note		64
Half note		32
Quarter note		16
Eighth note		8
Sixteenth note		4
Thirty-second note		2
Sixty-fourth note		1

Notes Detection Over Time

In order to create a real-time notes detector and writer, we must find a way to decide whether the new sample just recorded is a note, and if it is – does it simply the same continuous note from the previous sample?



Without any notes detection over time, the program output looks like that – whenever the new sample recorded is detected as a note, its been added to the music sheet. This of course is not how the result should be like.

From testing recording without notes detection over time, we can figure out the following:

- When a note is played, even for half a second, at least 3 samples detect this note.
- Noise does not appear to be a significant factor – the notes sample turned to be accurate even under noise conditions.

Giving those conclusions in consideration, We use the following method in our program:

- Record a buffer from microphone
- perform FFT and extract the strongest frequency
- Only if this frequency matches to a Recorder (instrument) note:
 - check the previous recorder note. If the same, update the note duration.
 - If not, a new note is being played.

Its important to say that other implementation as FIFO of the last notes and calculating average of last notes has been tested, but the method above gave the most accurate result.

ABC Music Notation

In this application, we chose the ABC music notation format in order to represent the input and output music sheets. This is because its simplicity (a plain text), and its many features.

The knowledge about the ABC notation is taken from the site:

- <http://abcnotation.com/>

Example of ABC file – contains from header, and notes

```
X:1
T:Notes
M:C
L:1/4
K:C
C, D, E, F, | G, A, B, C | D E F G | A B c d | e f g a | b c' d' e' | f' g' a' b' | ]
```

Notes



ABC as input

In the application, we use an internal Note class that saves the information on each note. We've wrote a parser to parse the ABC file into list of Note objects.

ABC as output

After recording, the user is able to save the recorded notes – as ABC, postScript or PDF.

In order to convert from ABC file to PDF:

1. ABC to PostScript – via the open source program **abcm2ps** - <http://moinejf.free.fr>
2. PostScript to PDF – via the open source utility **GhostScript** - <http://www.ghostscript.com>

Technical Problems & Solutions

Recording While Displaying

The main idea behind the project is to be able to record notes and see them appear in real-time on the music sheet on the screen, one by one.

These two operations (displaying on screen & recording), are both complex by themselves, and take a great deal of resources (memory, CPU) from the computer.

The first design approach was multi-threaded. The main idea was to separate between the recording and drawing operations, making them independent of one another.

Both operations would be inside of a loop.

The Recording loop would sample the Waveform audio input device (Microphone), and fill up the queue buffer.

The drawing loop would poll the queue and draw the note, whenever one is ready.

The first problem that rose from this approach, was that having two threads constantly using up a noticeable amount of CPU time, plus saving a queue of notes, took a great deal of resources.

The second problem was having to maintain synchronization primitives to synchronize between the recording and drawing operations, which also took up resources.

These problems tipped the scale towards a simpler approach, more procedural in nature. The approach that was chosen, was to have both the recording and the drawing in the same loop.

A note would be recorded, then sent directly to be drawn, and then back to the start.

The downside of this approach is that while the note is being drawn, the recorder may not try to record audio.

But because the recorder waits for the Waveform audio input to fill up, which happens independently of the program's flow, this downside is insignificant.

Displaying & Updating Music Sheet in Real-Time

The main idea behind the project is to be able to record notes and see them appear in real-time.

The main feature of Notey, displaying notes on-screen in real-time, required a solution which was effective and flexible.

The first solution that was checked, was to use a chain of 3rd party programs to convert each note to to a graphical representation.

After each note was played, it was added to a buffer of notes written in ABC notation.

The ABC buffer was then converted to postscript file format.

The postscript file was then converted to a PDF file, and then finally displayed on screen, using an Adobe controller.

This solution was abandoned, because of the high overhead it took to perform all these conversions, and also because displaying the PDF controller was very CPU intensive.

In order to save resources, the solution that was ultimately chosen was to draw the notes independently, using windows' GDI+ library.

Recorder Hero feature - implementation

The Recorder Hero feature of Notey allows for testing one's self in playing a series of notes.

After opening an ABC file and starting the recorder, the ABC data will appear under the Recorder Hero tab.

A marker will iterate on all of the notes, consecutively. The iteration speed can be decided by the user, using a scroll bar in the bottom of the screen.

Upon playing the correct note, the displayed note will turn green.

If the note's duration has passed, the note will turn red.

In order to be able to have a functionality of stalling some time on each note, while still being able to receive messages about notes played from the user, a multi-threaded approach is needed.

An iterator thread iterates on the notes, and sleeps for the duration dictated by the note. The main thread sends messages about notes played, and so if a note is played that matches the current note to be played, it can turn green immediately.

All the iterator thread needs to do, is to check whether the correct note has been played when the sleep times out. If not, it turns the note red, and carries on to the next note.

Project QA

Note: the application musical quality assurance was made most of the time by given application for musicians for testing.

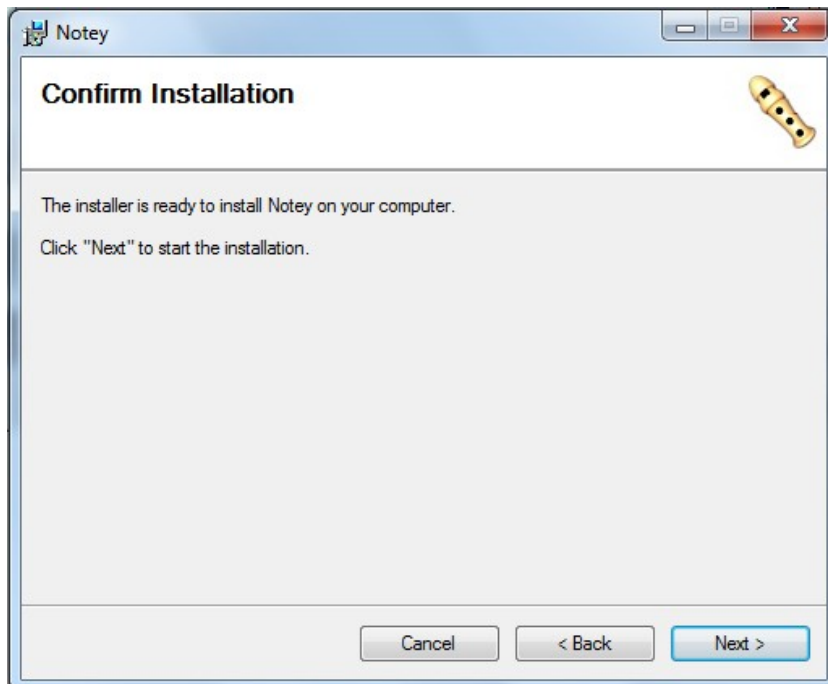
Type of Test	Test Description	Predicted Result	Succeeded / Failed
Notes recognition	Play notes C..A and see if the program detects them	Full detection	succeeded
Pitch recognition	Run a python script that plays frequencies	Full detection	succeeded
Save record as WAV	Saving the last note in wave format	File saved and open successfully	succeeded
Save record as WAV with illegal name	Save file with title and composer contains not alphabetical characters	'Save As' dialog shows an error message	succeeded
Save record as ABC	Saving the record as ABC, and check if consistent with the recording	The ABC output matches the music played	succeeded
Save record as PS	Saving the record as PS	PS output is correct and matches the music played	succeeded
Save record as PDF	Saving the record as PDF	PDF output is correct and matches the music played	succeeded
Save record as PDF with 'bad' name	Save record as PDF with extra long title & composer name	PDF output is correct and matches the music played	succeeded
Play the last record as WAV	Play the last record in the output sound device	Hearing the record	succeeded
Play the last record as synthesized WAV	Play the last notes as synthesized WAVE in the output sound device	Hearing the synthesized notes	succeeded
Draw a new note played	Play a different note from the last one	The note is detected and drawn	succeeded
Draw the previous note played after a short break	Play the same note twice with a tiny break	The note is detected and drawn twice	succeeded
Ignore noise	Shout and make frequencies not in the recorder range	The noise is ignored and not detected as notes	succeeded
Open a new recording while recording	Open new recording while the recorder is working	A new music sheet appears	succeeded
Open a new recording while recording is off	Open new recording	A new music sheet appears	succeeded

Open an ABC file	Open an ABC file	The ABC file is parsed as intended and displayed in music sheet	succeeded
RecorderHero on	Open an ABC file and press on the start record button	The note to be played appear in blue on music sheet and the iteration starts	succeeded
Open an invalid ABC file	Open a file that is not an ABC file	The parser ignore the other notes and parse it like an ABC file	succeeded
Speed up and down recorderHero	Speed recorderHero to max speed	The notes 'running' on screen, smoothly	succeeded
Resizeing window when in RecorderHero tab	Change the window size and see whether the new drawn music sheet is correct	The new music sheet is correct and the current note remains the same	succeeded
Resizeing window when in Notes tab	Change the window size	The music sheet stays correct	succeeded
Check other recorders	Open recorder music (on youtube). Record it, and see the notes appear on screen.	The program recognize those notes	succeeded

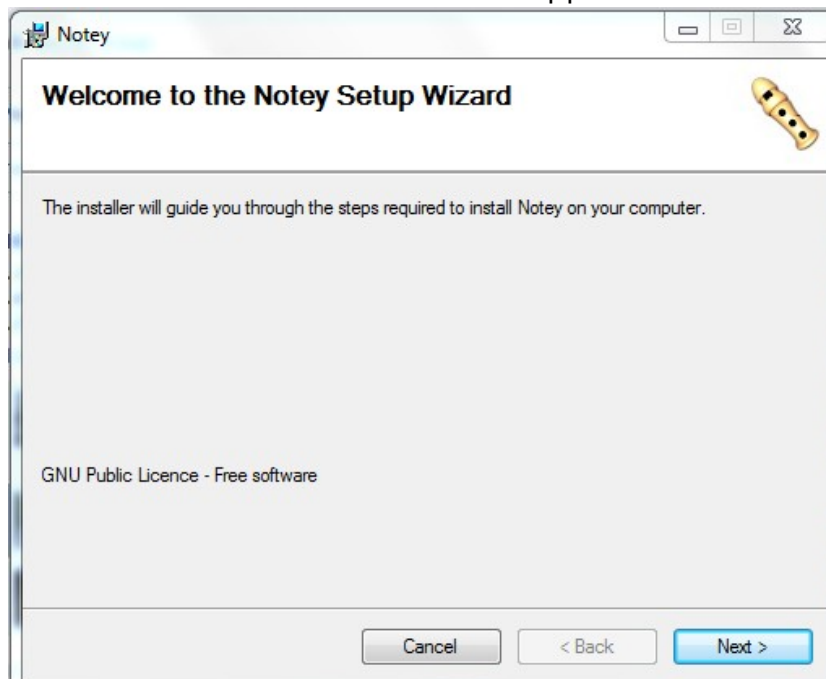
Application User Guide

Installation

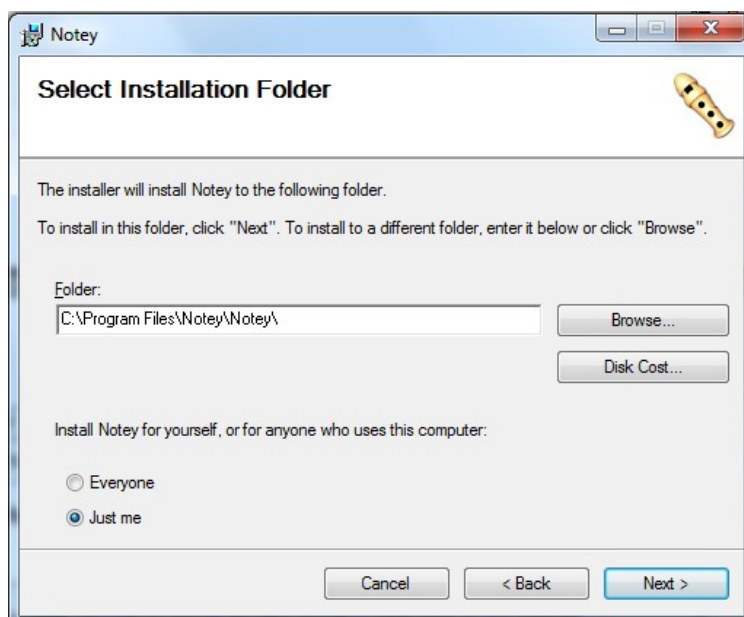
1. Click on the Notey installation file.
2. The 'Welcome' window will appear. Click 'Next'.



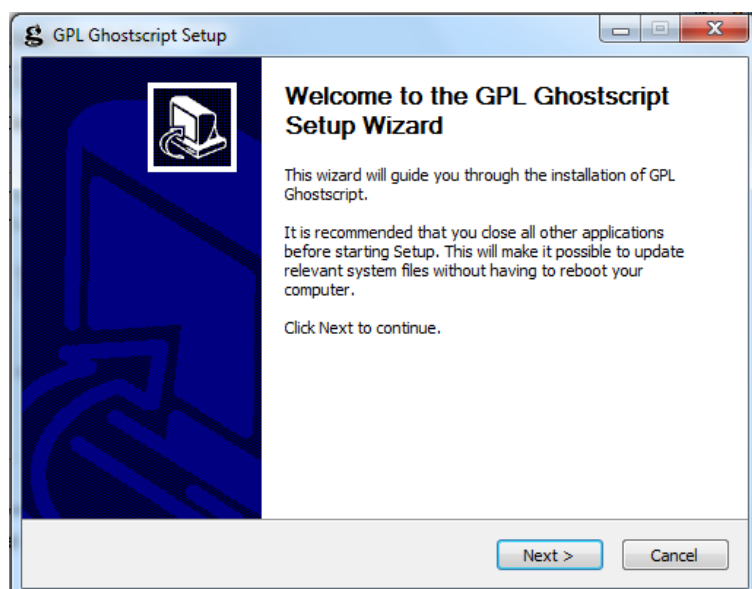
3. The 'Confirm Installation' window will appear. Click 'Next'.



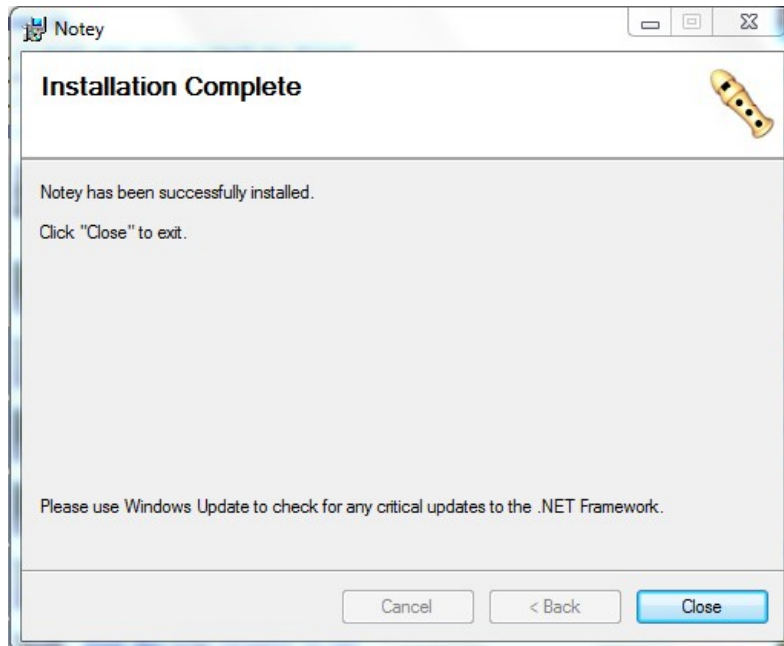
4. The 'Installation Folder' window will appear. It is advised to do not change the installation folder.



5. The installation should begin.
6. The GhostScript installation should begin. Install it in the default folder (advised).



7. The installation is complete.



Installation Requirements

Make sure you have installed on your computer the basic installation requirements:

1. DOT NET 4.0
2. GhostScript
3. abcm2ps
4. PDF reader
5. Make sure your computer has a working waveform input device (microphone), and an audio output device.

Setting Configuration

In order to use Notey correctly, you must set some configurations first.
The configuration window will appear at the first time you use Notey.

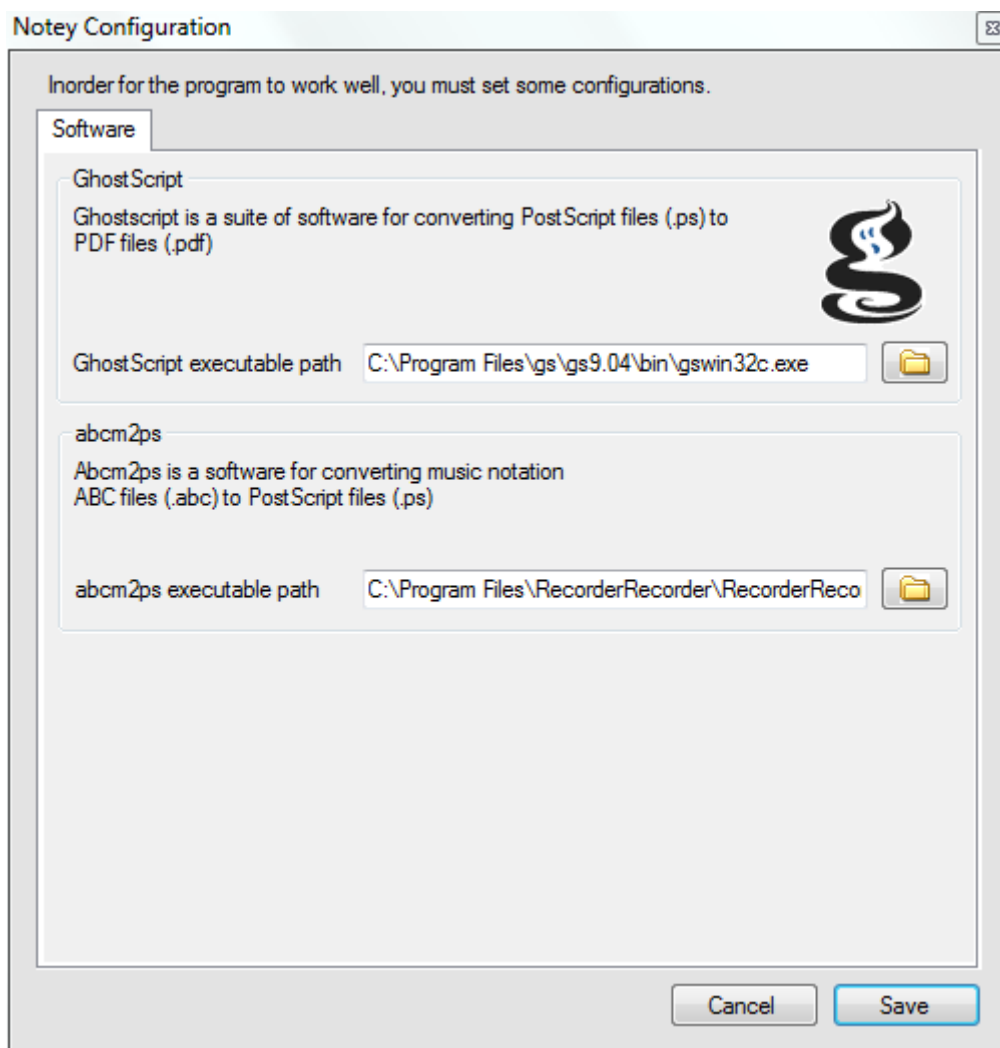
You can open and change settings by going to:

Main menu → Tools → Configuration

Alternatively, you can press:

Alt+T, Alt+C

The following window will appear:



Setting GhostScript

GhostScript is a suite of software for converting PostScript files (.ps) to PDF files (.pdf).

In the program, It's used for generating a music sheet.

Notey installs the GhostScript software, but in case you might want to use a different version of GhostScript installed on you computer, you must provide Notey the GhostScript executable path.

The GhostScript executable path is usually at the following location:

C:\Program Files\gs\gs9.04\bin\gswin32c.exe



GhostScript version

To configure the GhostScript executable path:

- At the Configuration window (See: [Setting Configuration](#)), at the tab 'Software', go to the 'GhostScript' panel.
- Click on the directory Icon. Go to the GhostScript executable path and select it.

Setting ABCM2PS

ABCM2PS is a software for converting ABC music notation files (.abc) to PostScript files (.ps).

The ABCM2PS software is included inside the Notey program folder, but in case you might want to use a different version of abcm2ps, you must provide Notey the abcm2ps executable path.

The abcm2ps executable path is usually at the following location:

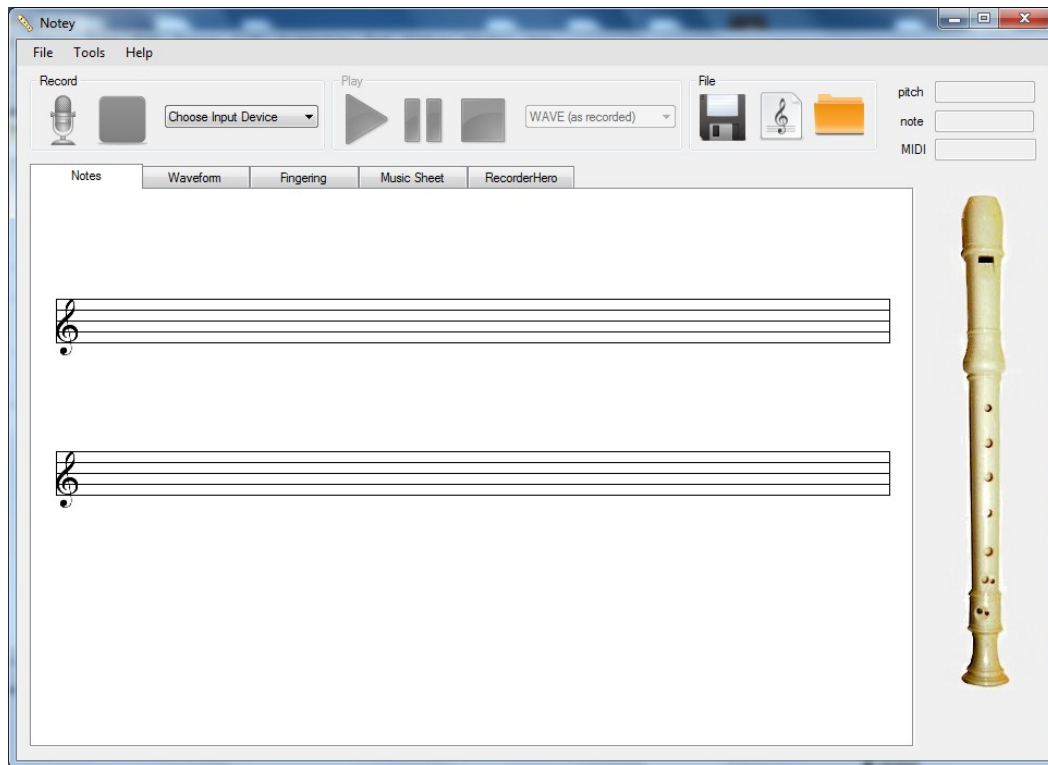
C:\Program Files\Notey\Notey\abcm2ps-5.9.24\abcm2ps.exe

To configure the ABCM2PS executable path:

- At the Configuration window (See: [Setting Configuration](#)), at the tab 'Software', go to the 'abcm2ps' panel.
- Click on the directory Icon. Go to the abcm2ps executable path and select it.

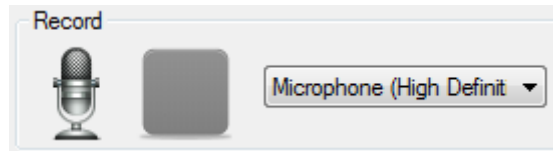
Recording

With Notey you can record yourself playing on the Recorder and see the notes appear on the music sheet.



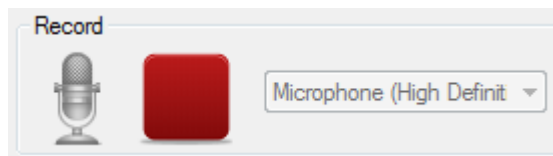
Start Recording

- Go to the "Record" panel.
- Chose Input Device – chose the recording microphone from the list.
- The Recording Button will be enabled (you will be able to click on it) – Click to start the recording.



Stop Recording

- After starting a recording, Go to the "Record" panel.
- Click on the Stop Button.

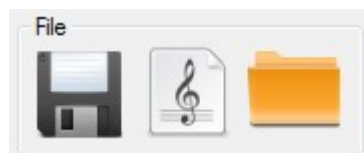


New Recording

You can restart your recording at any time (whether if the program is currently recording or not). This action will remove the old notes you've played.

You can do this either way:

- Click on the New Music Sheet button, on the "File" panel.



- Go to: ***Main menu → File → New***
- Press: ***Ctrl+N***

Real-Time Recorded Notes Music Sheet

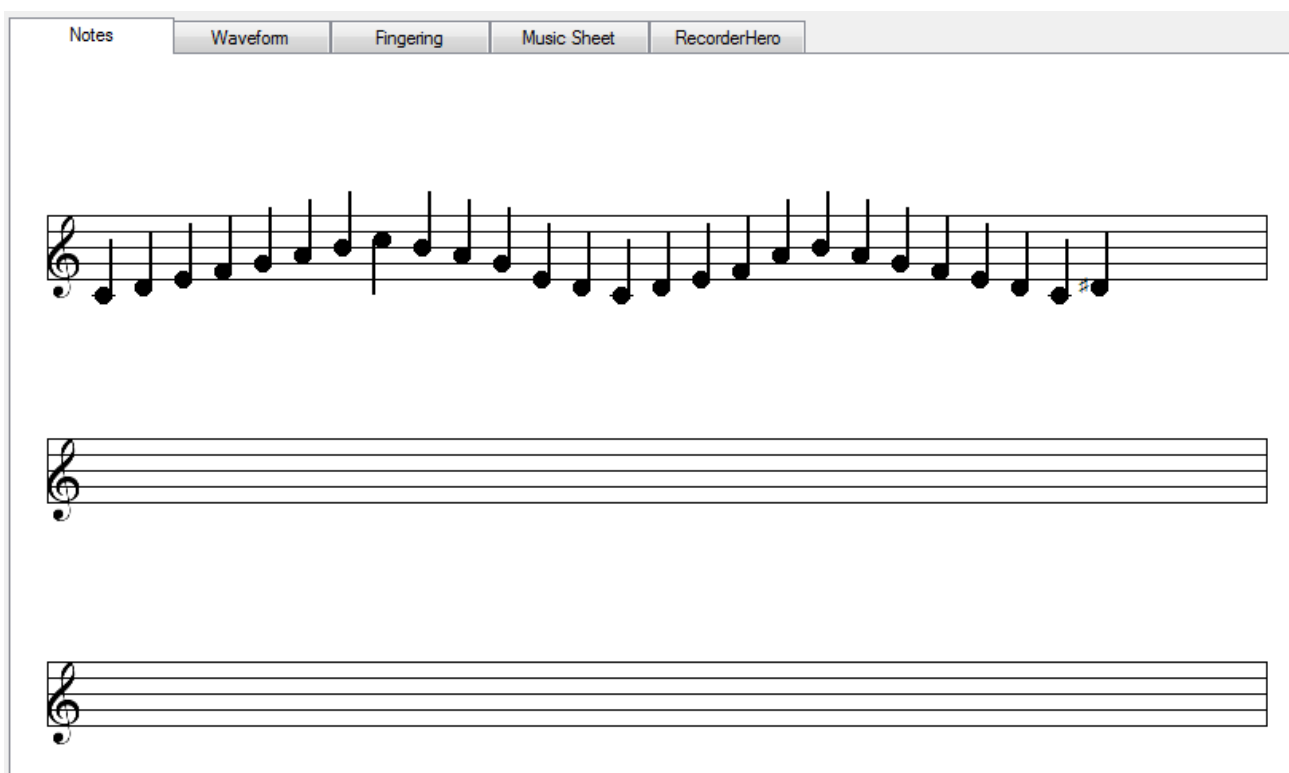
While in [Recording Mode](#), the 'Notes' tab displays the notes played. When a note is played for amount of time (at least to be consider as a note and not noise), it will be added to the real-time music sheet.

When not in Recording Mode (after pressing stop), the notes will remain.

When the music sheet is filled, a blank music sheet will appear.

To open the 'Notes' tab:

- Simply click on the 'Notes' tab



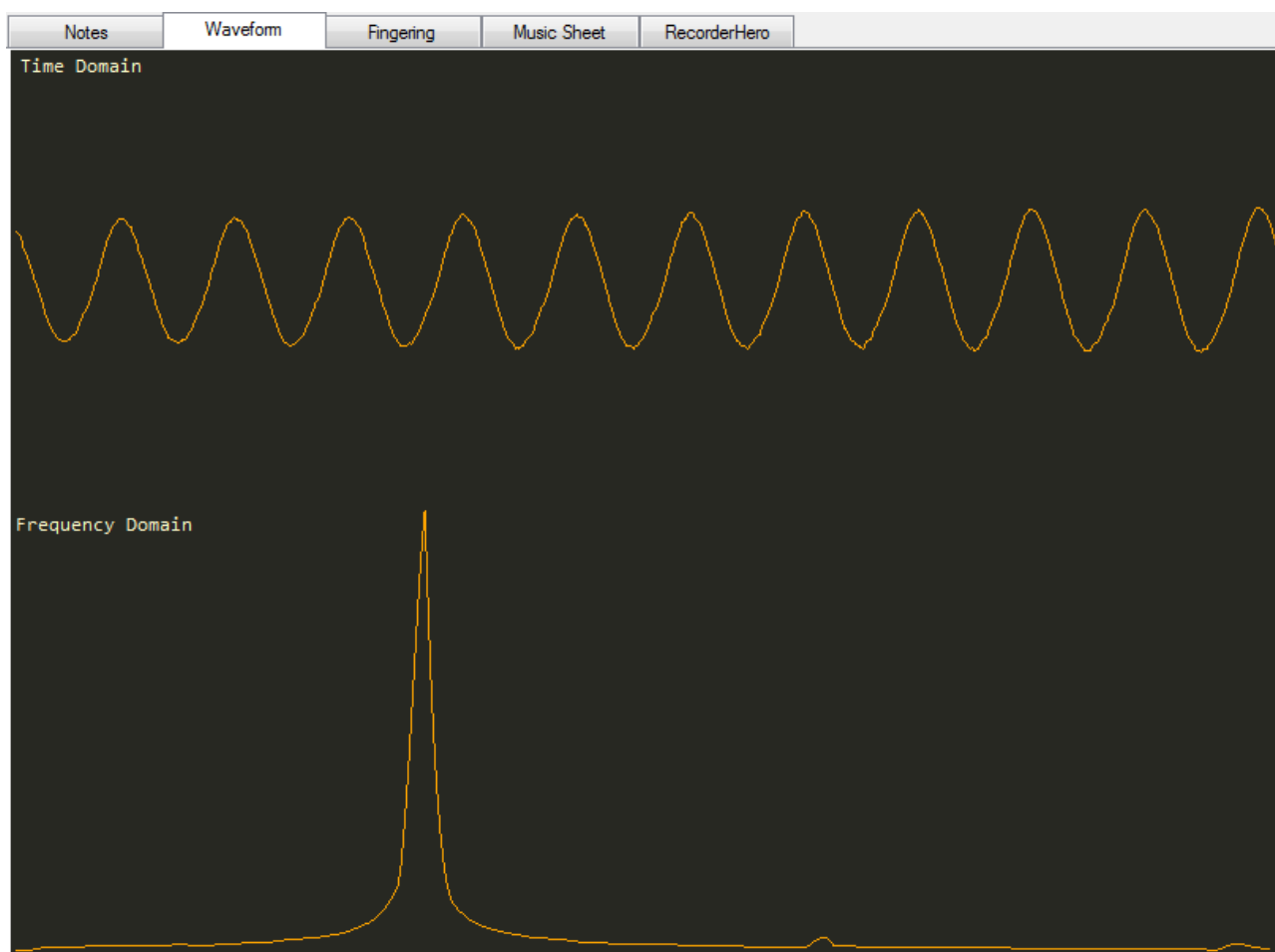
The screenshot shows the RecorderHero application interface. At the top, there is a tab bar with four tabs: 'Notes', 'Waveform', 'Fingering', and 'RecorderHero'. The 'Notes' tab is currently selected. Below the tab bar, there are three musical staves. The top staff contains a sequence of notes, including quarter notes, eighth notes, and a sharp sign. The middle and bottom staves are empty.

Real-Time WaveForm

While in [Recording Mode](#), the 'Waveform' tab displays the current sound waveform and it's frequency analysis.

To open the 'Waveform' tab:

- Simply click on the 'Waveform' tab



PDF Music Sheet

The 'Music Sheet' tab present the notes played so far in a .pdf file.

This tab is only available if you have a PDF reader installed on your computer.

You can set the music sheet Title and Composer, by simply filling the textboxes in strip.

To see the music sheet:

- Click on the tab 'Music Sheet'
- Enter the wanted music sheet's title and composer name in the matching textbox – this is optional.
- Click on the 'Refresh' button.
- Right-click on the music sheet will open a context menu strip of the PDF reader installed on your computer. In it you will usually find options like saving the file, printing it, changing the display preferences and etc.



Note Data and Fingering

In the right side of the program's main window, there is the 'Note Data and Fingering' panel.

For every note, the following information is presented:

- Pitch – the frequency of the note (in Hertz)
- Note – the note name
- MIDI – the MIDI number of the note
- Fingering picture – the fingering of the note (how to hold the Recorder instrument in order to play the note).

While in [Recording Mode](#), if the software detects that a musical note has just been played, the note's fingering and data is displayed.

While using the [RecorderHero](#), data and fingering for the current note to be played is displayed, so you can see how to play that note.



Playing

After recording has [finished](#), the recorded notes can be played using the player in the 'Play' panel.

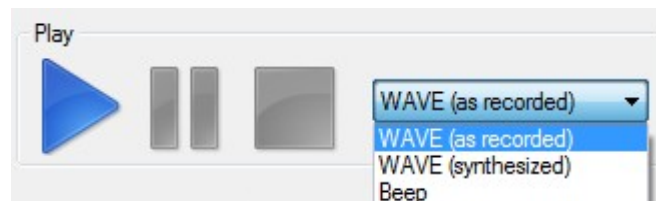


The 'Play' panel contains 3 buttons and a list:

- **Play** – start playing the recording
- **Pause** – pause playing the recording
- **Stop** – stop playing the recording

From the **list**, you can chose the format of recording you want to play:

- **WAVE (as recorded)** – play the real record from the last record session, with



all the noise.

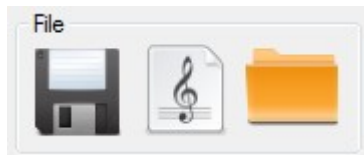
- **WAVE (synthesized)** – play the notes recorded in a synthesized manner
- **Beep** – plays the notes recorder as beeps from the computer speaker.

Saving a Recording

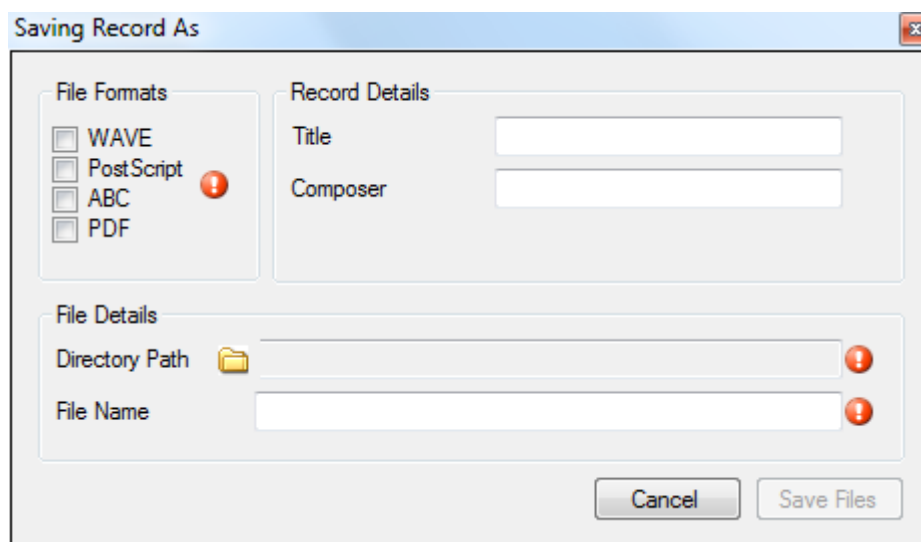
At any point in time you can save the recorded notes you've played.

You can do it in several ways:

- Click on the 'Save' button, on the "File" panel.



- Go to: **Main menu → File → Save**
- press: **Ctrl+S**



On the 'Saving Record As' dialog, fill the following fields:

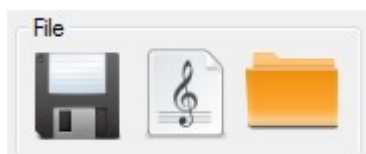
- File Formats – the saved record format,
- Record Details – the record title and composer. Optional.
- File Details – where to save the record.

Open Music Notes File (RecorderHero)

The Recorder Hero feature of Notey allows for testing one's self in playing a series of notes.

open an ABC file

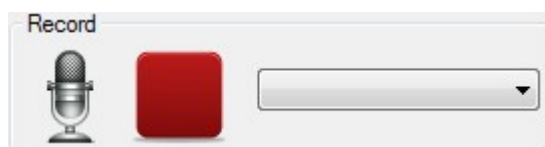
- Click on the 'Open' button, in the 'File' panel.



- Go to: **Main menu → File → Open**
- Press: **Ctrl+N**

Start and Pause the RecorderHero

Use the 'Start Recording' and 'Stop Recording' buttons. Select a microphone and start playing (see [Start Recording](#)). In order to start over, re-open the file.



Speed Up/Down the RecorderHero

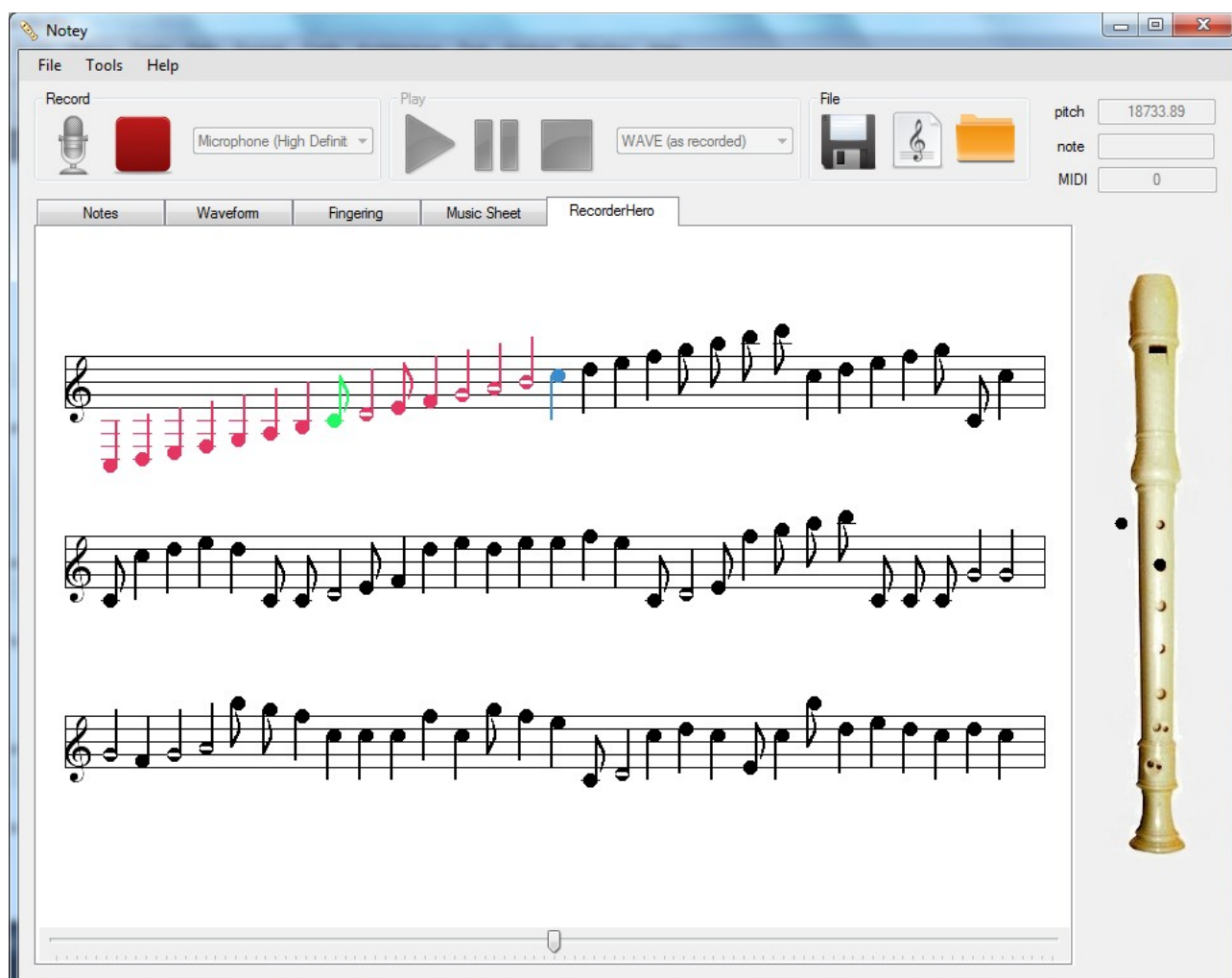
You can speed up or down the RecorderHero speed (if you feel that the notes notes' playing tempo is too fast/slow for you).

Go to the track bar in the bottom of the RecorderHero tab, and move the cursor right (to speed up) or left (to slow down).

Using the RecorderHero

- First, open the 'RecorderHero' tab by clicking on it (if it is not already open).
- Press on the recorder button (if it is not already pressed).
- After opening the ABC file, that music notes will appear in the music sheet. Each note will have its own color:
 - **Blue** – the note that should be played right now.
 - **Red** – the last note will turn red if you haven't played it correctly.
 - **Green** – if the note that should be played right now (painted in blue) is played by your recorder (and the program recognize it), it will turn green.
 - **Black** – the notes that should be played next.

For each note that should be played right now, you can see its fingerings. You can also speed up/down the RecorderHero speed, and learn more easily how to play in the recoder by simply looking at the fingering of the note that should be played.

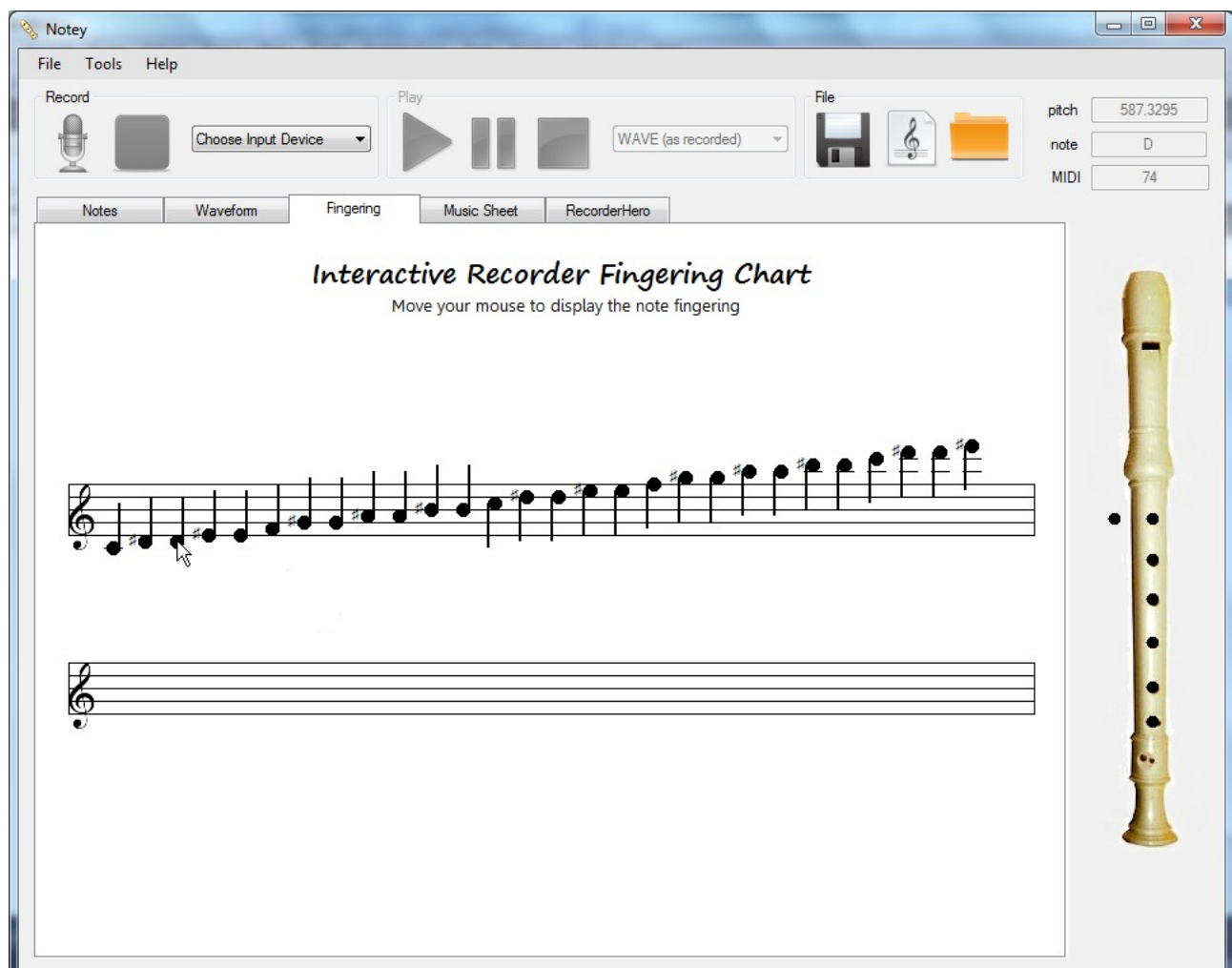


Interactive Music Sheet

While not in Recording Mode (after pressing stop), the interactive music sheet displays all the recorder's notes and their matching fingering.

Moving the cursor over a specific note will show its fingering in the recorder picture on the right side of the program window, and the notes' data (pitch, name, MIDI) in the top right note data panel.

For example, while moving the cursor over the note D, we can see the note's fingerings, and information:



Application Developer Guide

This part is meant to be read by anyone who intends to do some changes/improvements to the source code.

We'll start with a general overview of the system architecture and continue with more detailed specification.

Note: This guide is not going to cover all the components in the system. This is not its purpose. This guide is more of a 'Step by Step' guide for going through the program's main algorithms and architecture and then, if needed, the important data types, classes & etc. are to be explained.

For the full documentation and the source code, view the documentation created by doxygen (available in the application website).

Compilation Instruction & Requirements

- Open the project in Visual Studio. For creating the program, Visual studio 2010 was used.
- Make sure you have installed on your computer the basic installation requirements:
 - DOT NET 4
 - GhostScript
 - abcm2ps
 - PDF reader
- Make sure your computer has a working waveform input device (microphone), and an audio output device.

Program Main Flow

This program is a win32 .NET windows forms application. It's based on the event-driven programming approach.

The main thing thing you must keep in mind is that since it's a GUI program, all the calculations and continuous operations must be exported and executed in a different thread.

The program's "brain" and logic lies within the working threads. We will start by explaining their operations.

Recording thread

MainForm.cs: `private void RecordingLoop()`

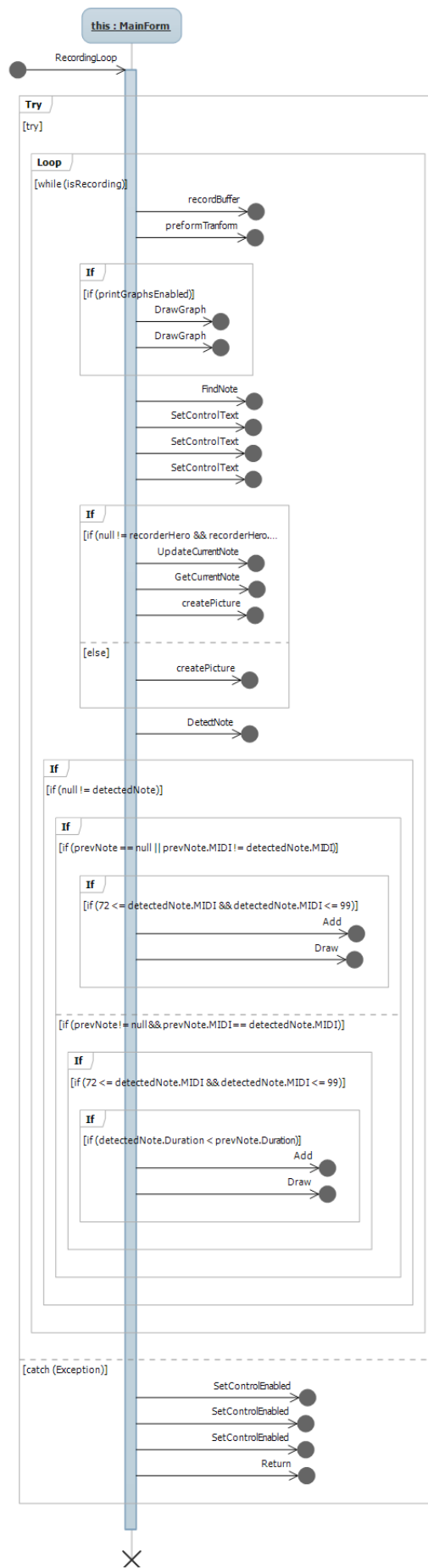
Note: first read the ['Recording while Displaying'](#) page for a better understanding.

After pressing the the 'Start Record' button, this thread starts executing a loop, which ends once the user has pressed the 'Stop Recording' button.

This thread is responsible of performing the following in a loop:

1. Record a sound waveform sample.
2. Analyze the sample.
3. Detect whether the sample is a note (previous, new or noise).
4. Display note on screen:
 - Update the 'Notes' tab - Add to music sheet and refresh.
 - Update the 'Waveform' tab - Display the time & frequency domains.
 - Update the fingering picture and note data in notes panel.
 - Update the 'RecordHero' tab - if the RecorderHero is working.

We'll walk through the thread's main function calls (as displayed in the UML chart below), and each time we shall explain about the components that are involved.



1. Record a Sound Waveform Sample:

```
WaveIn.cs: public void recordBuffer(byte[] waveArray, int sleepTime)
```

The **WaveIn** class is a C# wrapper for the win32 waveform audio API. The waveform audio API provides an application with exact control over waveform audio input/output devices.

The wrapper contains the basic functionality the program needs:

- Get a list of input devices (microphones)
- Start recording
- Stop recording
- Get a buffer that has just recorded – Receive a buffer with samples that was just recorded.

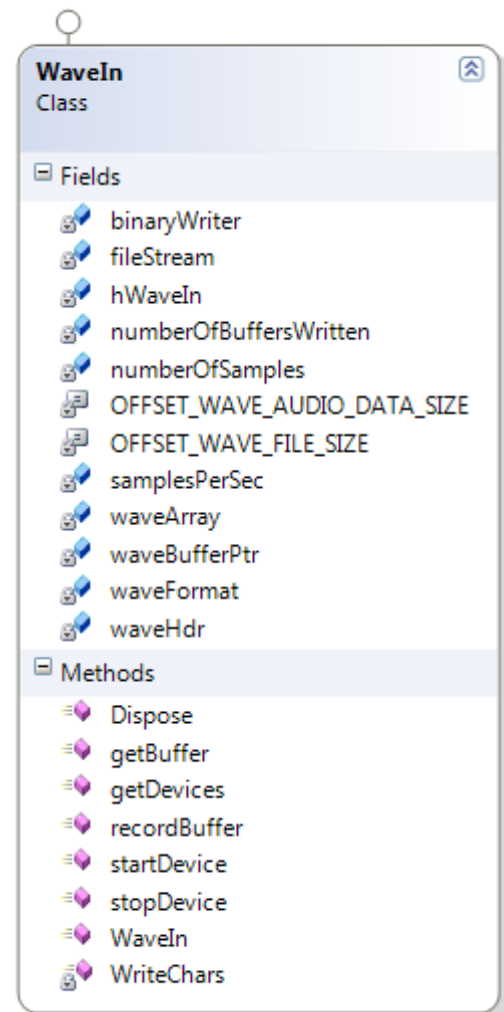
In the program, in the **MainForm** class we create an instance of **WaveIn**, with:

```
const uint NumberOfSamples = 4096;
```

```
const uint SamplesPerSec = 88200;
```

```
WaveIn.cs: public MainForm() {...
```

```
Wave = new WaveIn(NumberOfSamples, SamplesPerSec); ...}
```



Whenever we want to access the input waveform audio device, we do it with the **Wave** instance of **WaveIn**. This encapsulates the unnecessary implementation details of accessing the API and leaves us with a nice simple API.

2. Analyze the sample

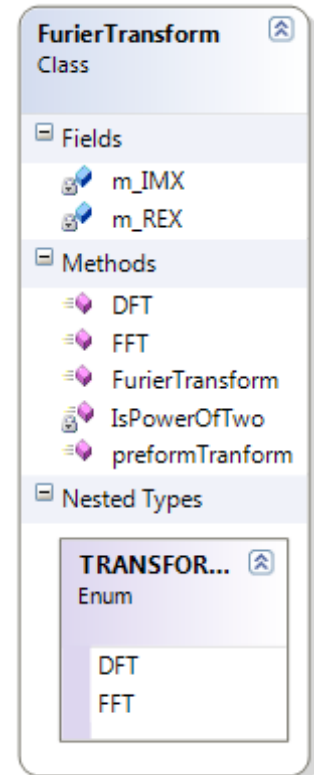
```
Analysis.cs: public double performTranform(TRANSFORM transform, byte[]  
inputBuffer, double[] outputBuffer, uint numberOfSamples, uint samplesPerSec)
```

The **FourierTransform** class is a C# class that provides implementations of the discrete fourier transform (DFT) and the fast fouier transform (FFT).

After sampling a buffer of waveform sound, we pass those samples to the function **performTransform**, that take its input (the samples, a.k.a 'Time Domain') and returns the result in the output buffer – the frequency domain, in the polar notation (see the [FFT chapter](#)). For each frequency in hertz, we can see its amplitude in the array.

The function **performTransform** can call the **DFT()** or the **FFT()** implementations – depends on the input **TRANSFORM**.

The return value of the transform is the max pitch detected.



3. Detect whether the sample is a note (previous, new or noise)

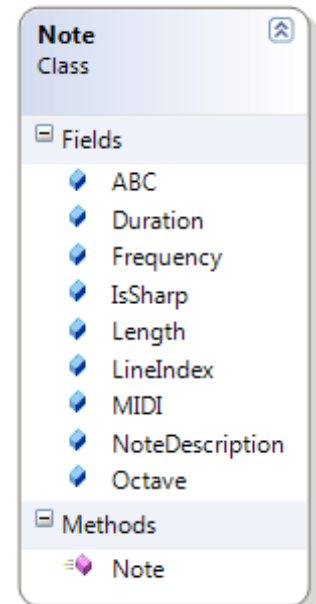
After getting a maximum pitch from the Fourier transform analysis, we want to know:

1. Does the following pitch indeed match a pitch of a known note, or was it just a noise?
2. If the pitch describes a note, is it the last note, or was a new note played?

First, let's see the structure of a **Note** class.

It contains information about a note:

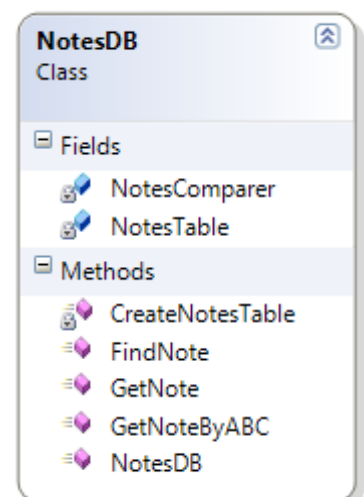
- ABC – abc notation of the note
- Duration – the duration in milliseconds of the note
- Frequency – the base frequency of the note
- IsSharp – is the note sharp (or bemol)
- Length – $\frac{3}{4}$, $\frac{6}{8}$, ...
- LineIndex – an interval variable, for drawing notes in sheet
- MIDI – the MIDI number of the note
- NoteDescription – "A", "B", "C#", ...
- Octave - (-1)..9



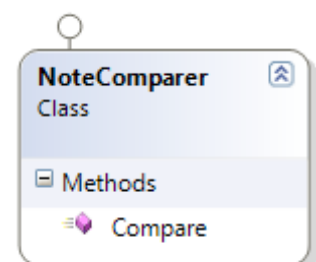
We create at the beginning of the program a **Note Data Base**, the **NotesDB** class.

That NotesDB is a simple list of **Notes**, that holds the notes, and their information, and provides a binary search of a note by its frequency (this is done by using the NoteComparer):

```
NotesDB.cs: public Note FindNote(float frequency)
```



So, we now, given the pitch extracted by the FFT from the sampled buffer, we can see if it's a note or a noise (null is returned).



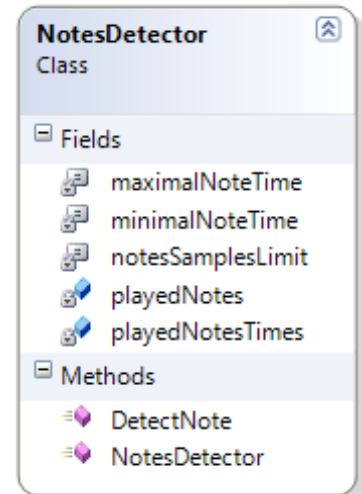
All that remains is to detect whether the note just found (if any) is a new note, another occurrence of the last note, or – the last note played again.

```
NotesDetector.cs: public Note DetectNote(Note note, int timePlayed)
```

The **NotesDetector** class implements the notes detection over time. It holds a list of the last notes played, and in what time.

When the function **DetectNote** is called, it looks back on the last notes played and their time – if it seems that the note is a new note, and it has at least 3 or more samples with the same frequency – then the note is returned. Else, a null is returned.

For the implementation in details you can read the source code.



4. Display Note on Screen

The last step in the recordingLoop thread is displaying the new recognized note on the screen. This means, we should update it in several different places in the main form. This part is a GUI programming.

Using Delegates

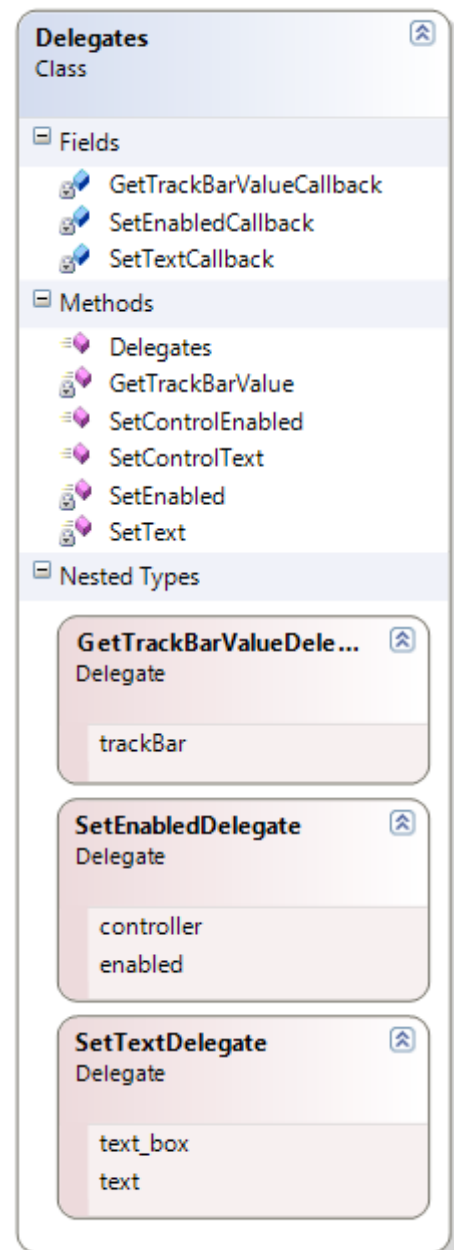
If you are making a GUI application and you are using multiple threads, there is one very important rule: **GUI controls can only be accessed from the GUI thread.** This is inherent to Windows development and is not a limitation of .NET.

So, when we want to access the GUI from the recordingLoop thread, we must do it in a different way.

This is done by creating a wrapper class for the “invoke” calls – instead that we will have this implementation for each controller in the main form, we export it into a different class.

Read more about invoking a controller from a different thread in the MSDN article: [How to: Make Thread-Safe Calls to Windows Forms Controls](http://msdn.microsoft.com/en-us/library/ms171728%28v=vs.80%29.aspx) -

<http://msdn.microsoft.com/en-us/library/ms171728%28v=vs.80%29.aspx>



Update the 'Notes' tab - Add to Music Sheet and Refresh:

Given the last note played, if it's a new one, we want to add it to the real-time music sheet.

The **NoteDrawer** class provides us a simple interface for creating a music sheet picture and adding notes to it.

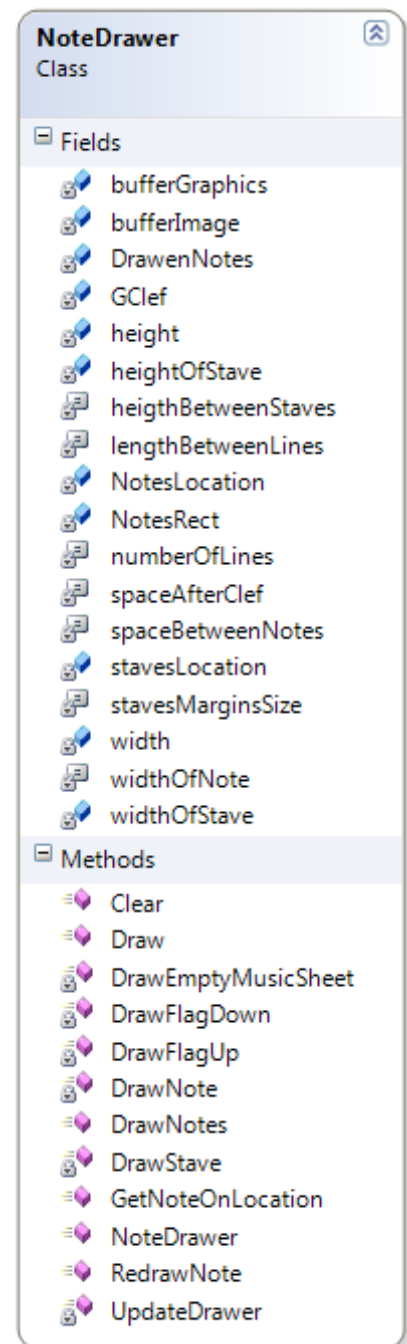
```
public int Draw(Graphics graphics, Color background, Brush brushSheet, Brush  
brushNote , Note note = null)
```

The NoteDrawer class maintains a list of the last notes added. When the current music sheet page is filled, the NoteDrawer is responsible for clearing and drawing a new music sheet. So, the programmer who uses this object, don't have to worry whether the user has just resized the window so the music sheet size has changed.

The music sheet drawing is done by using simple Graphics functions, provided with the .NET graphics control.

When initialized, or when there is a need to re-paint (when the window change its size), the NoteDrawer calculates how many staves can be placed inside the new picture size, and their location. For each staves, the locations of the notes that can be placed on the staves is saved.

So, in run time, when a new note should be added to the music sheet, the NoteDrawer checks if it has place on current music sheet and places it there, else, it draws a new music sheet.



Update the 'Waveform' Tab - Display the Time & Frequency Domains:

The function for drawing the time domain and frequency domain (input & output buffer) is **DrawGraph**. It is a simple GUI function.

```
WaveGraphics.cs: static public void DrawGraph(...)
```

Update theFingering Picture and Note Data in Notes Panel:

Given the current note, we need at recording time to display its data in the note data panel: its frequency, name & MIDI.

Recalling the delegates methods, we use those functions in order to access the textBox controls in the main form.

```
public void SetControlText(System.Windows.Forms.Form form, Control control, string text)
```

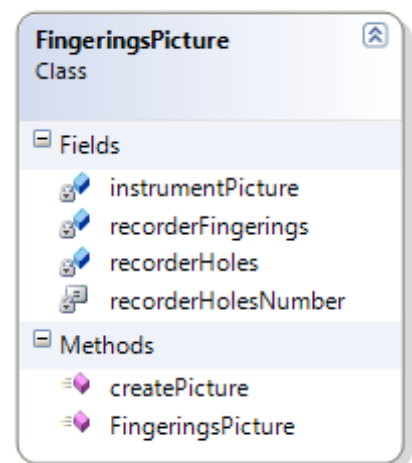
The recorder fingerings picture is created with the **FingeringsPicture** class.

```
FingeringPicture.cs: public void createPicture(Graphics graph, Note note)
```

The **createPicture** function gets a note, and print a picture of a recorder with the matching holes fingerings covered.

It uses two classes:

- **recorderFingering** – for each recorder note, save a list of the notes fingerings.
- **RecorderHoles** – the locations of the holes in the instrument picture.



The **createPicture** take the input note, and gets its fingerings from the **recorderFingering**, and draws the fingerings according to the **RecorderHoles** locations.

RecorderHero Thread

```
RecorderHero.cs: public RecorderHero(Graphics graphics, string abcFilePath)
```

Note: first read the ['RecordingHero feature'](#) page for a better understating.

In the constructor, the RecorderHero gets the Grpahics object (where to paint the interactive music sheet) and the ABC file contains the notes. It parses the file, and saves the notes and the rest of the recording data in the variable *AbcData*.

After pressing the 'Start Recording' button (or if the button already pressed), the RecorderHero starts, and starts executing a different thread:

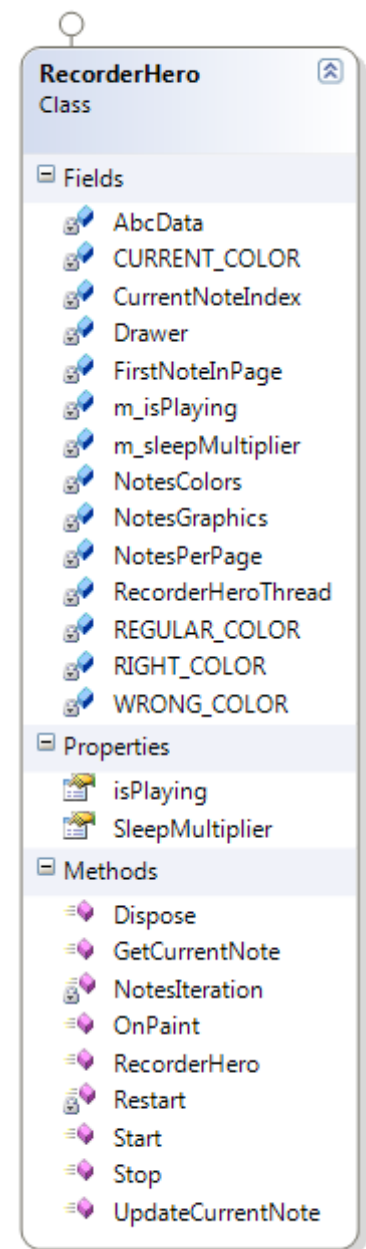
```
RecorderHero.cs: private void NotesIteration()
```

In this thread, there is a loop that iterate on the notes by their order. In pseudo code:

1. For each **note** in **notes**:
 1. Add this **note** to the music sheet. Draw it in blue.
 2. Sleep (**note**.duration * user_speed) – the note's duration is the time in milliseconds that the note should be played. The user_speed is how fast/slow that user want the appearance of the notes will be.
 3. If the note hasn't been played, draw it in red.

As you might notice, We're lack here that part of the integration between the ReorderHero and the RecordingLoop.

How can the RecorderHero tell whether the correct note has been played, and to redraw it in green?




```
RecorderHero.cs: public void UpdateCurrentNote(Note currentNote)
```

The function **UpdateCurrentNote** is called from the **RecordingLoop** thread, each time a note is detected. This function checks whether the current note that should be played (given by the **CurrentNoteIndex** variable) is the note recorded in the RecordingLoop, and if it is – draw it green.

This way, the logic behind the RecorderHero's note iteration is kept simple and easy to maintain, while the interface given to the RecordingLoop thread handles the note's comparing.