Disclaimer

Development of a Music Score Editor based on MusicXML

Najeeb Khan
Speech Signal Processing Lab
University of Ulsan

Overview

- XML
- MusicXML
- Music Score Editor

XML

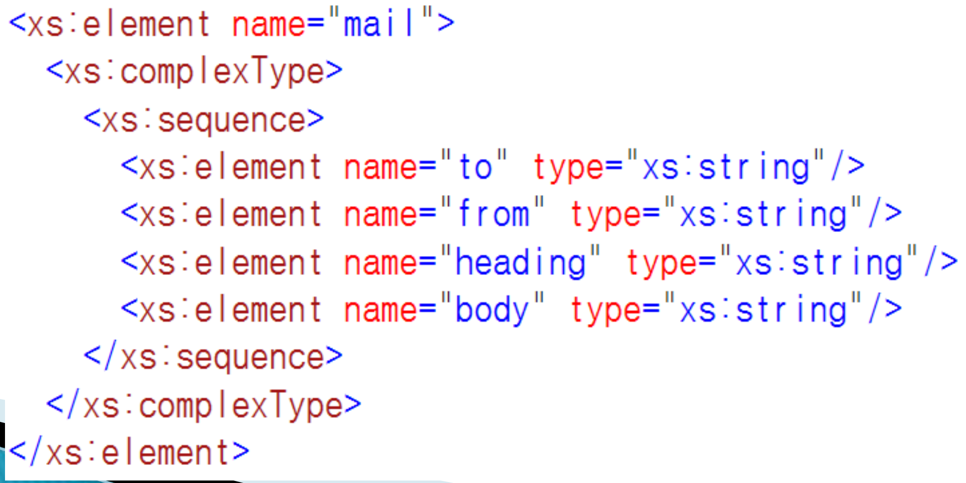
- XML was designed to transport and store data independent of software and hardware

```xml
<mail id="3">
     <to>Najeeb</to>
     <from>John</from>
     <heading>Reminder</heading>
     <body>Don't forget me this weekend!</body>
</mail>
```

- Well Formed XML follows the following rules
  - XML documents must have a root element
  - XML elements must have a closing tag
  - XML tags are case sensitive
  - XML elements must be properly nested
  - XML attribute values must be quoted

XML Contd…

- Valid XML documents must satisfy two conditions
  - It must be well formed
  - It must conform to a document type
- Document types can be specified in two ways
  - Document Type Definitions
  - XML Schema



```
<xs:element name="mail">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

MusicXML

- There are many fine computer music programs in the world. Unfortunately, sharing music between them used to be difficult
- MIDI is a wonderful format for performance applications like sequencers, but it is not so wonderful for other applications like music notation
- The goal is to create a universal format for common Western music notation

MusicXML

- Say you have 100 music applications, each with its own format
- For each application to communicate with the other, 10,000 separate programs would need to be written without a common interface language
- With a common interface language, each application writes only one program, so only 100 separate programs would be required

MusicXML

- Say you have 100 music applications, each with its own format
- For each application to communicate with the other, 10,000 separate programs would need to be written without a common interface language
- With a common interface language, each application writes only one program, so only 100 separate programs would be required

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

- <?xml version="1.0" encoding="UTF-8" standalone="no"?>
  - Setting the value of standalone to "no" means that we are defining the document with an external definition in another file

- <!DOCTYPE score-partwise PUBLIC
 "-//Recordare//DTD MusicXML 3.0 Partwise//EN"

 "http://www.musicxml.org/dtds/partwise.dtd">
    - We are using MusicXML, specifically a partwise score where measures are contained within parts. We use a PUBLIC declaration including an Internet location for the DTD

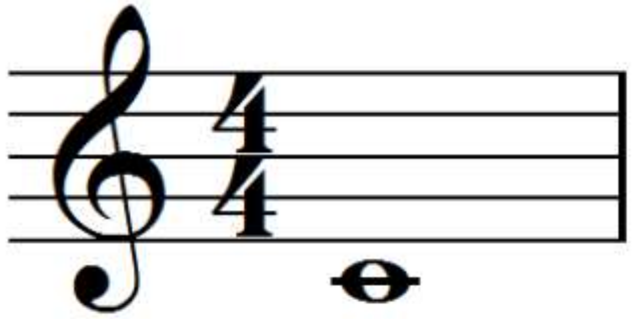
```
<part-list>
    <score-part id="P1">
        <part-name>Part 1</part-name>
    </score-part>
</part-list>
```

- A MusicXML file starts off with a header that lists the different musical parts in the score

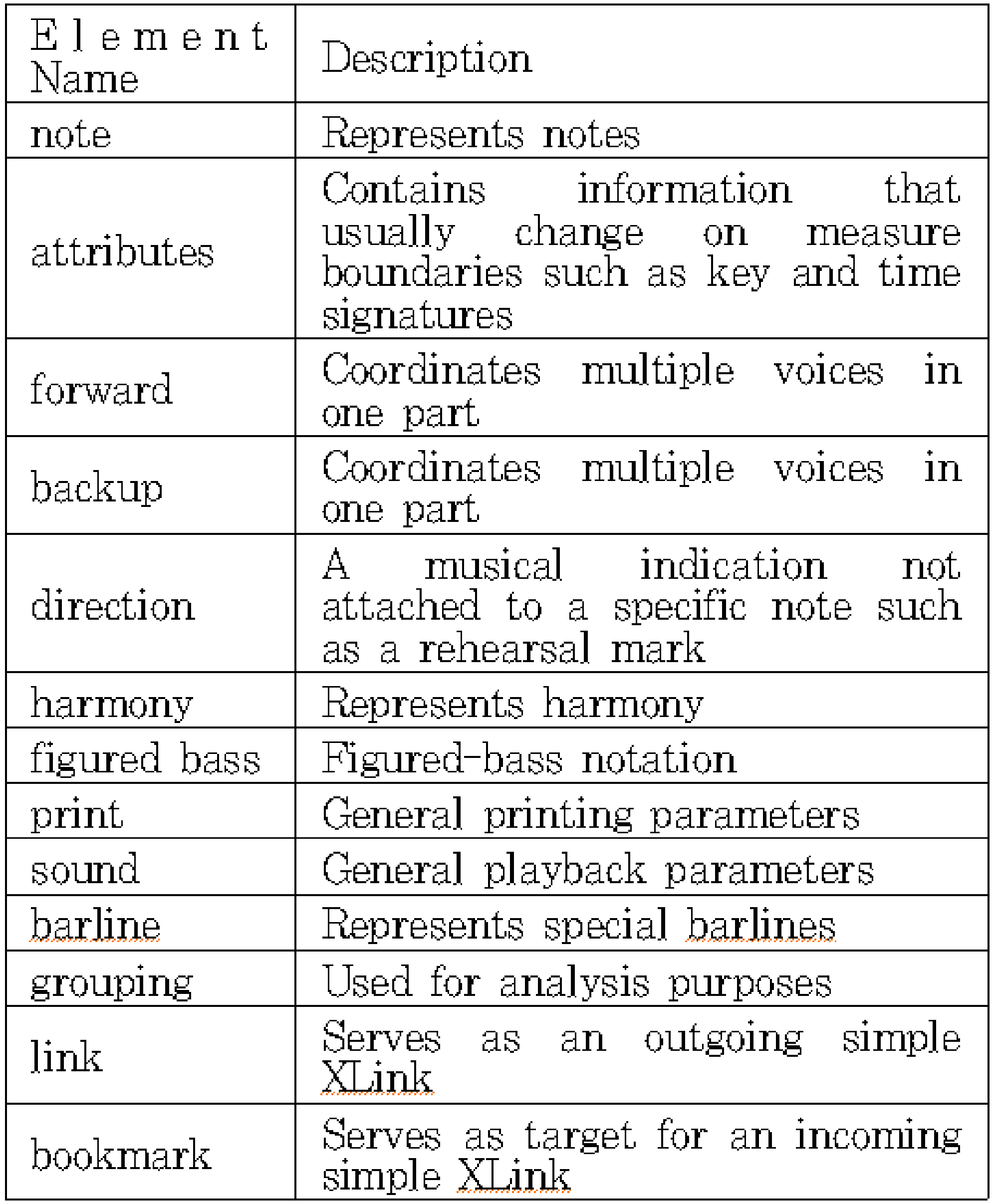
- <part id="P1">
  - beginning the first part within the document
- <measure number="1">
  - Starting the first measure in the first part
- <attributes>
  - <divisions>1</divisions>
  - <key>
      <fifths>0</fifths>
    </key>

```
<time>
     <beats>4</beats>
     <beat-type>4</beat-type>
 </time>
```

```
<note>
        <pitch>
                <step>C</step>
                <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
</note>
```

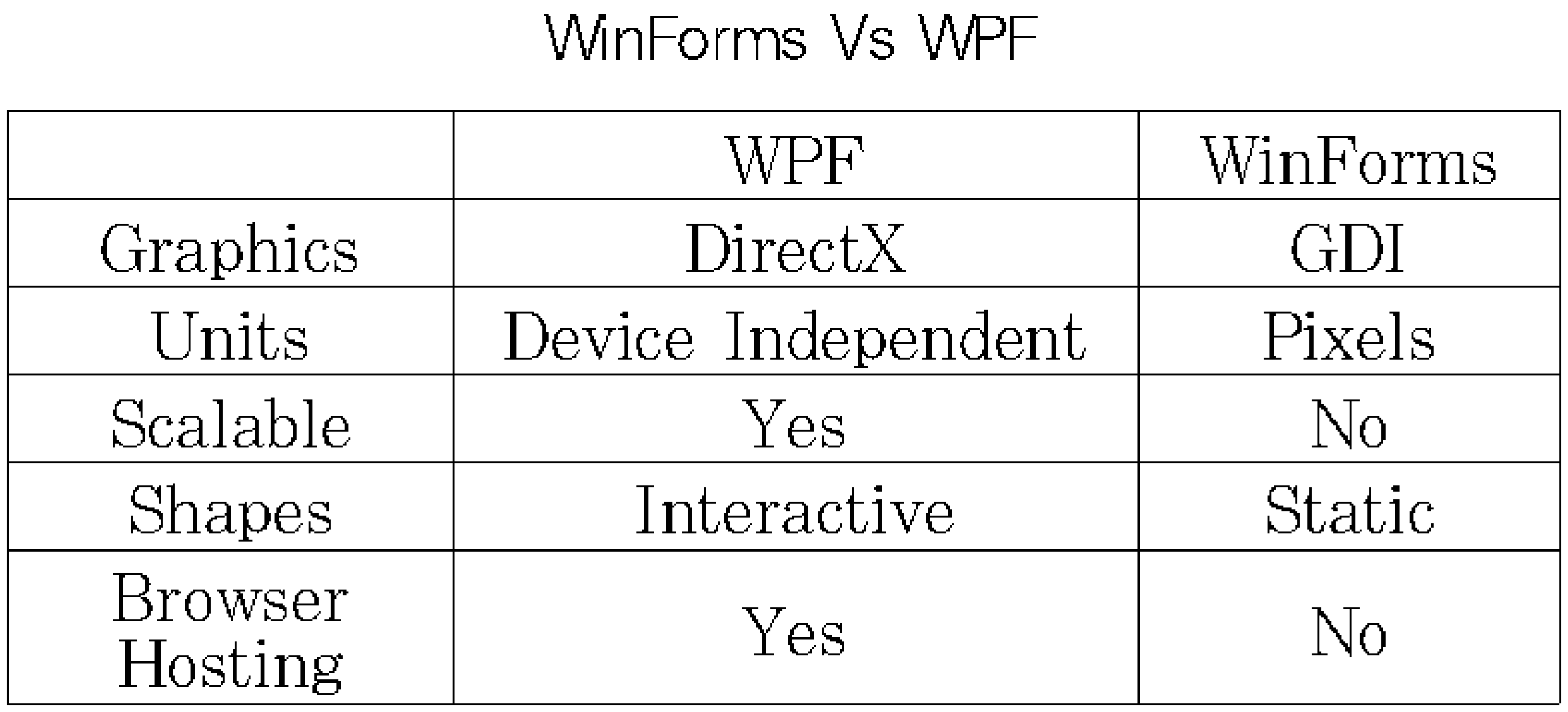


MusicXML Structure



Element Name	Description
note	Represents notes
attributes	Contains information that usually change on measure boundaries such as key and time signatures
forward	Coordinates multiple voices in one part
backup	Coordinates multiple voices in one part
direction	A musical indication not attached to a specific note such as a rehearsal mark
harmony	Represents harmony
figured bass	Figured-bass notation
print	General printing parameters
sound	General playback parameters
barline	Represents special barlines
grouping	Used for analysis purposes
link	Serves as an outgoing simple XLink
bookmark	Serves as target for an incoming simple XLink

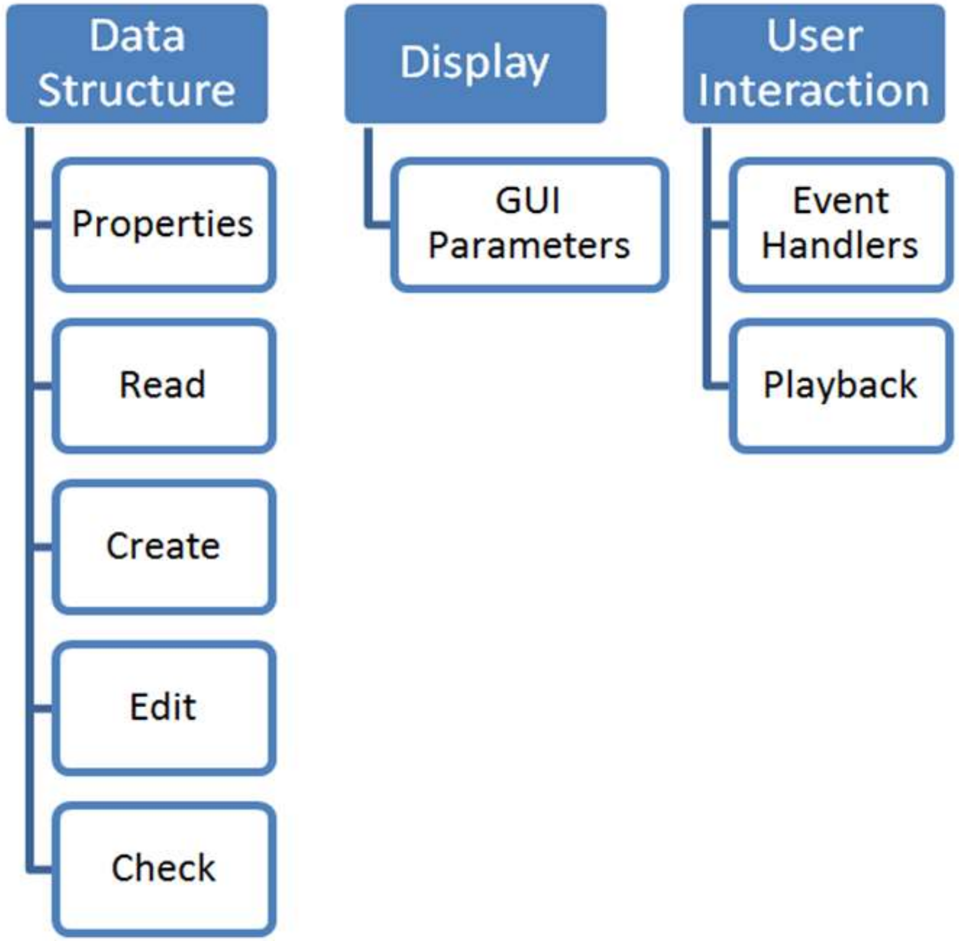
Music Score Editor



WinForms Vs WPF

	WPF	WinForms
Graphics	DirectX	GDI
Units	Device Independent	Pixels
Scalable	Yes	No
Shapes	Interactive	Static
Browser Hosting	Yes	No

Music Score Editor



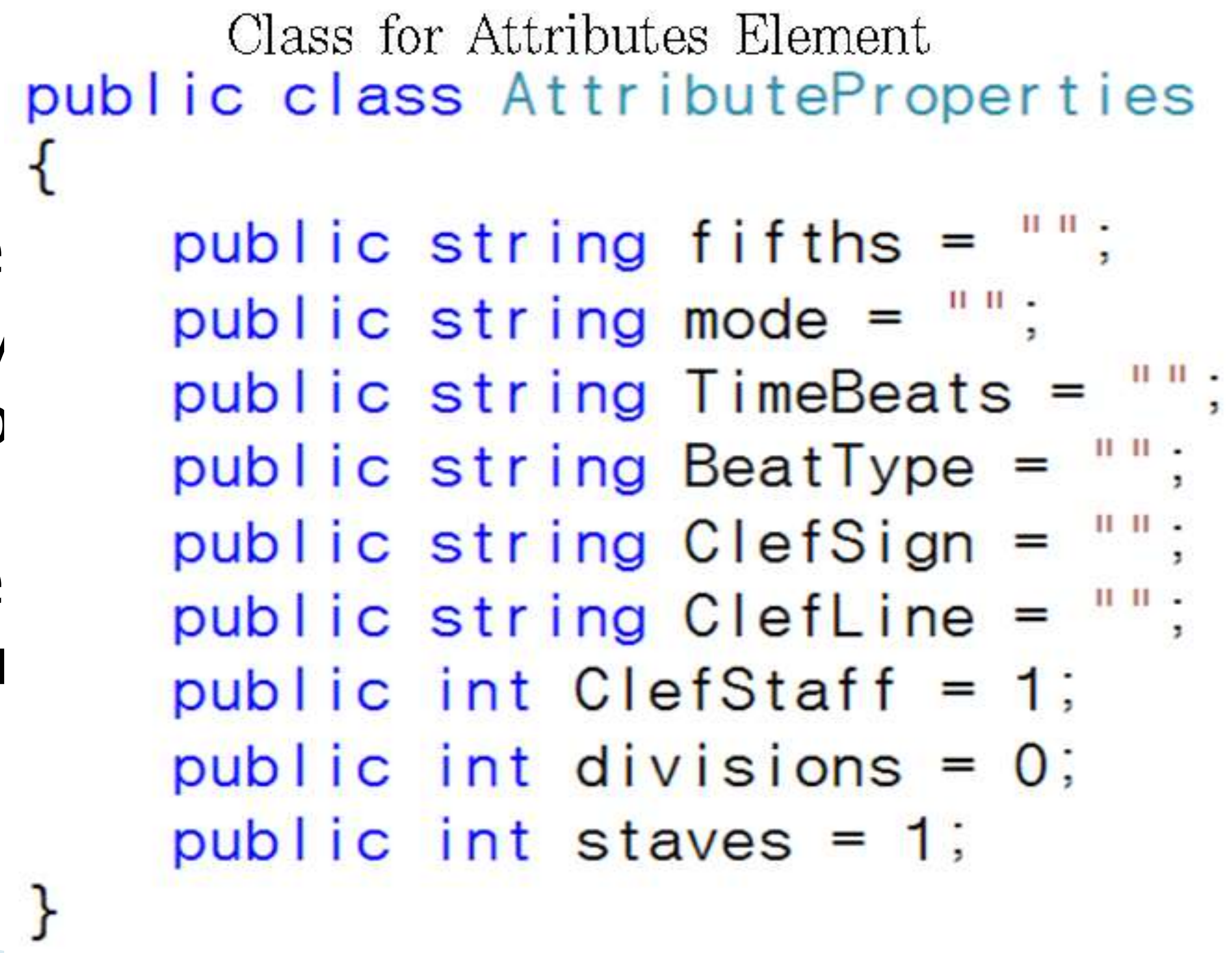
Music Score Editor Modules

Data Structure

```
//Access the first part in the score
ScorePartwise.Part FirstPart =
   (ScorePartwise.Part)scorePartwise.getPart().get(0);

//Access the first measure in the Part
ScorePartwise.Part.Measure FirstMeasure =
   (ScorePartwise.Part.Measure)FirstPart.getMeasure().get(0);

//Check the Music data inside the measure and look for the first note
Note FirstNote=null;
for(int i =0; i < FirstMeasure.getNoteOrBackupOrForward().size(); i++)
{
   if (FirstMeasure.getNoteOrBackupOrForward().get(i) is Note)
   {
      FirstNote = (Note)FirstMeasure.getNoteOrBackupOrForward().get(i);
      break;
   }

}
//Get the pitch of the note
string step = FirstNote.getPitch().getStep().value();
//Get the duration of the note
int duration = FirstNote.getDuration().intValue();
```

Data Structure

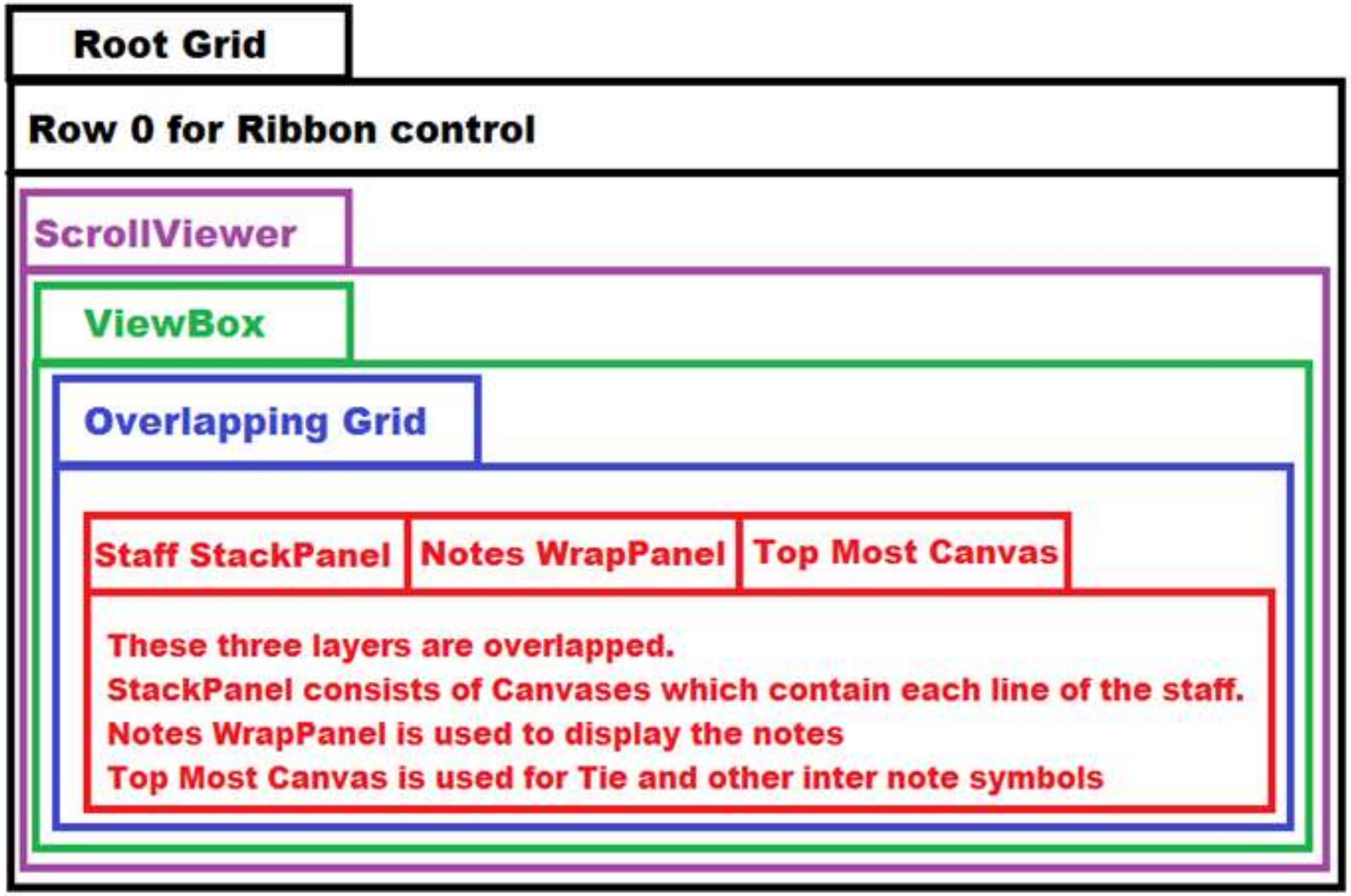
- It i rectly for ; and me
- Sev each rep s of ele the me and att



Class for Attributes Element

```
public class AttributeProperties
{
    public string fifths = "";
    public string mode = "";
    public string TimeBeats = "";
    public string BeatType = "";
    public string ClefSign = "";
    public string ClefLine = "";
    public int ClefStaff = 1;
    public int divisions = 0;
    public int staves = 1;
}
```

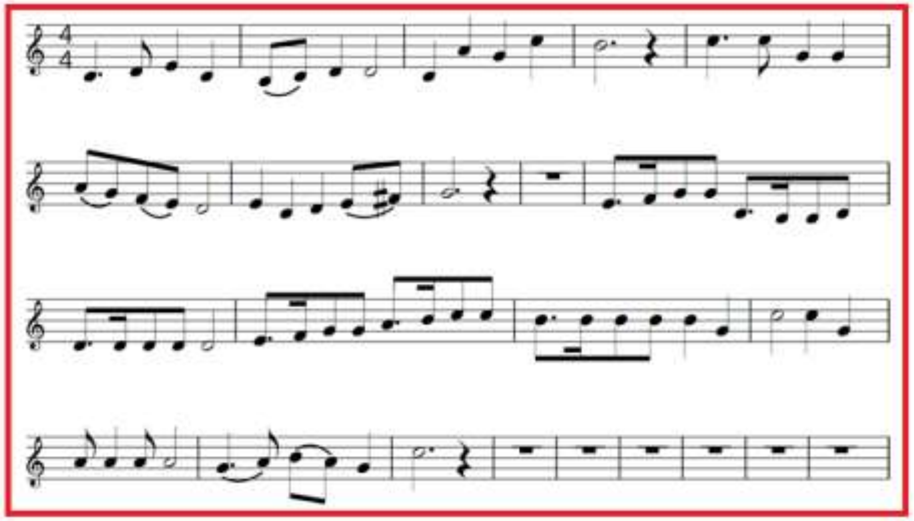
GUI Architecture



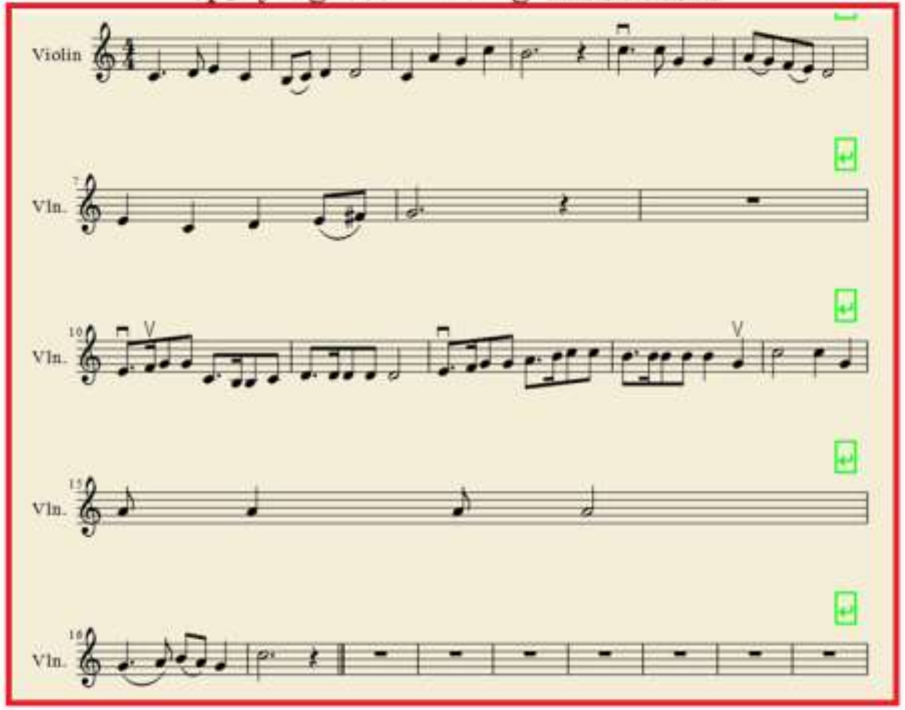
These three layers are overlapped.
StackPanel consists of Canvases which contain each line of the staff.
Notes WrapPanel is used to display the notes
Top Most Canvas is used for Tie and other inter note symbols

Results

Displaying score using the music score editor

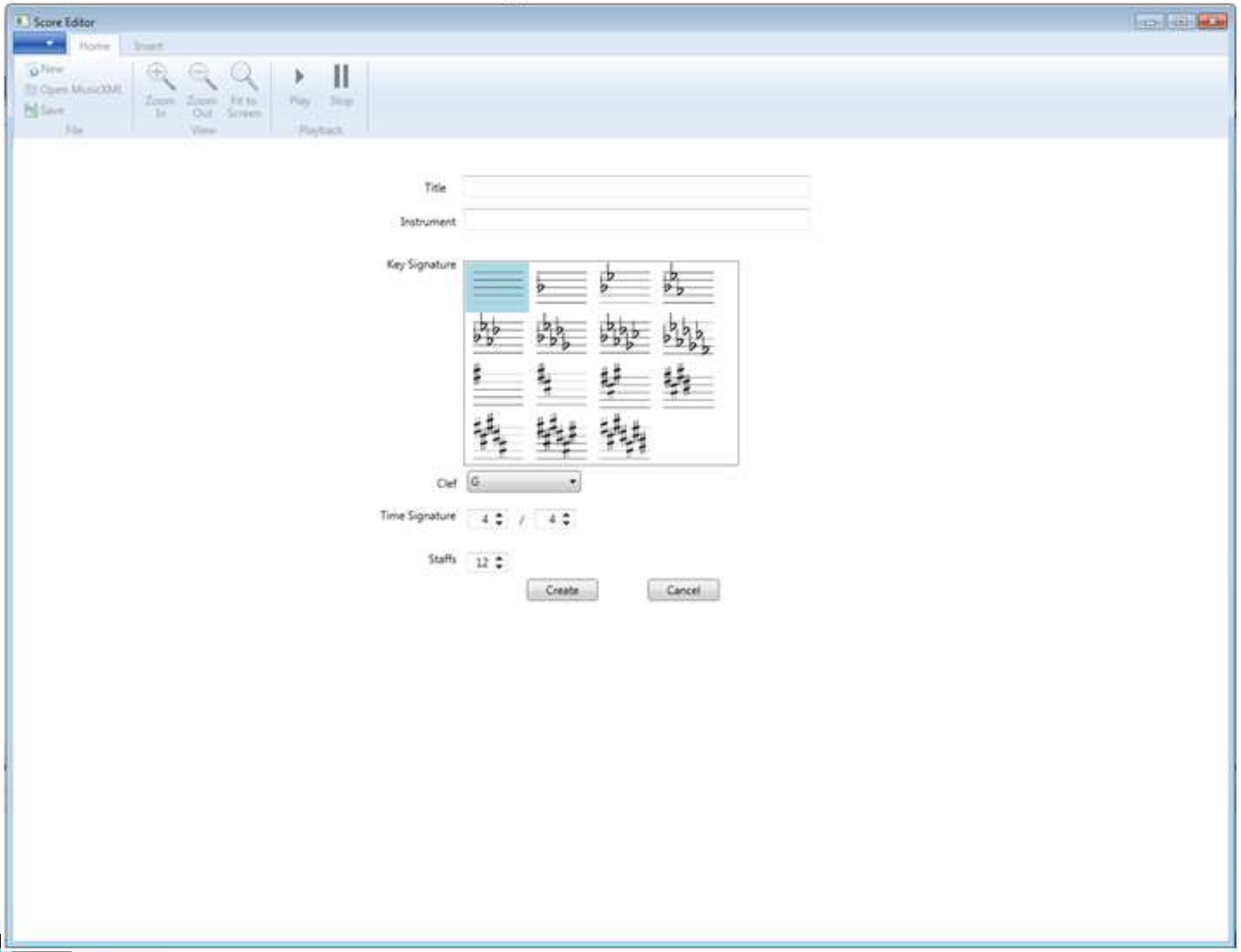


Displaying score using MuseScore



Results

Creating a new score



Title

Instrument

Key Signature

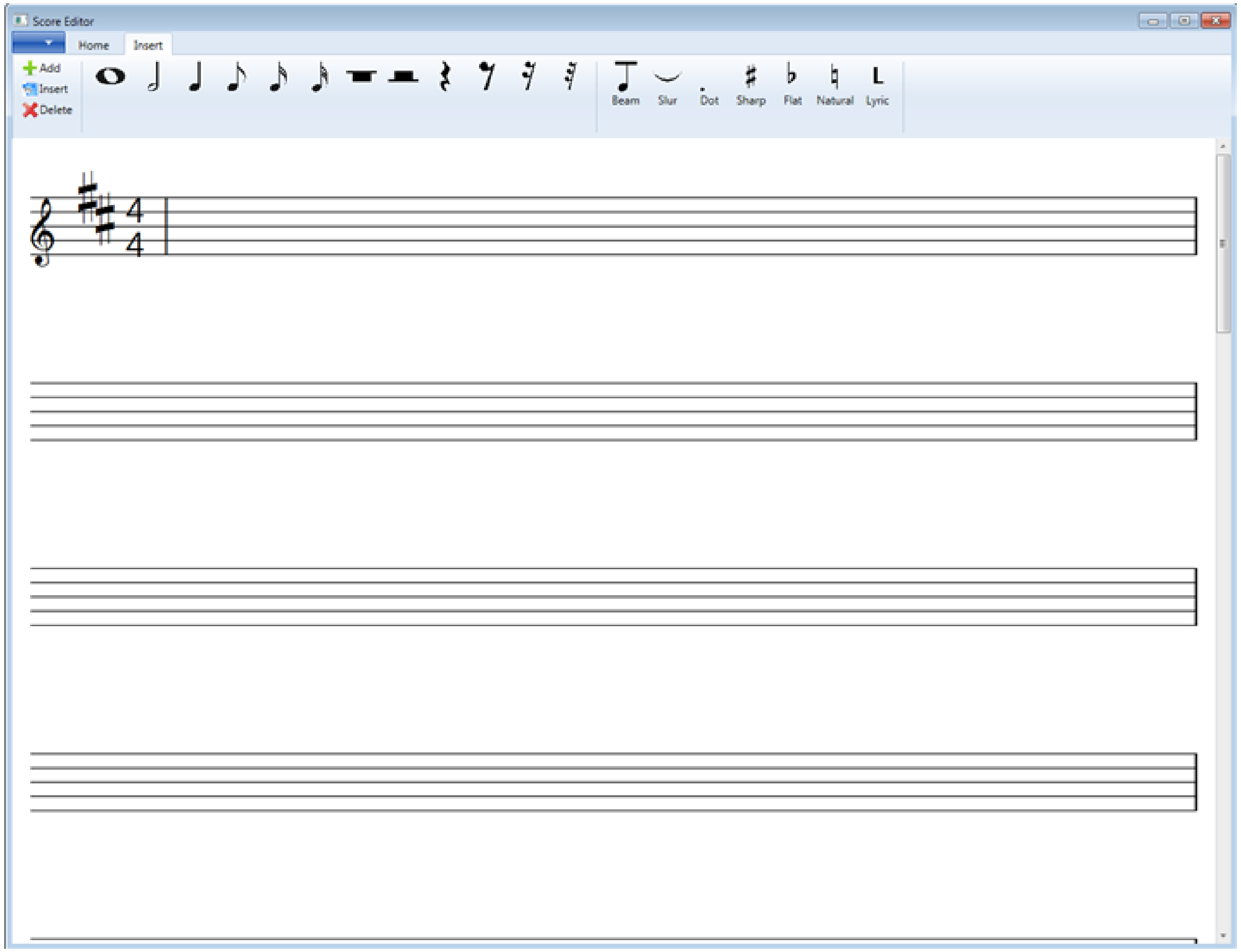
Clef G

Time Signature 4 / 4

Staffs 12

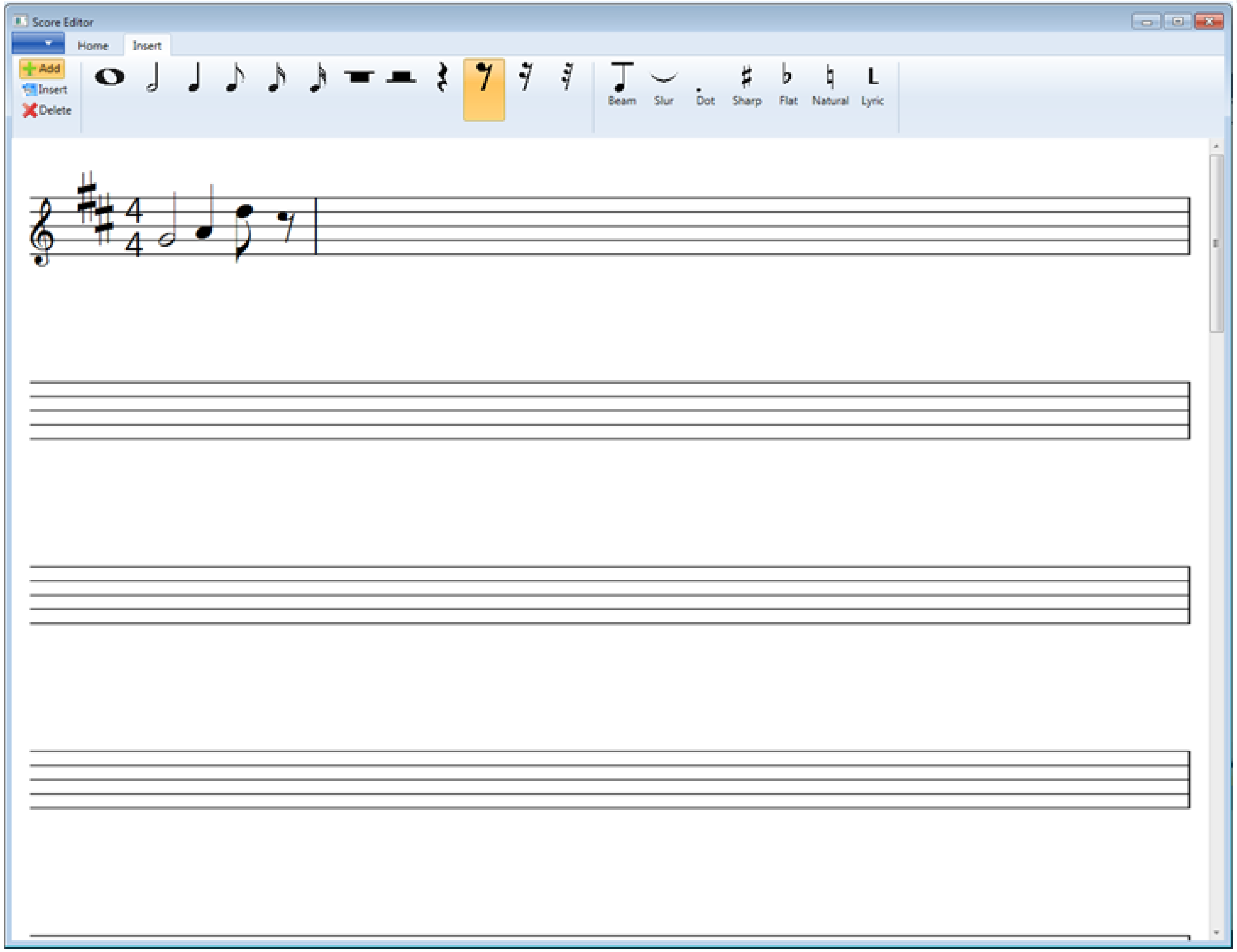
Results

A new score displayed

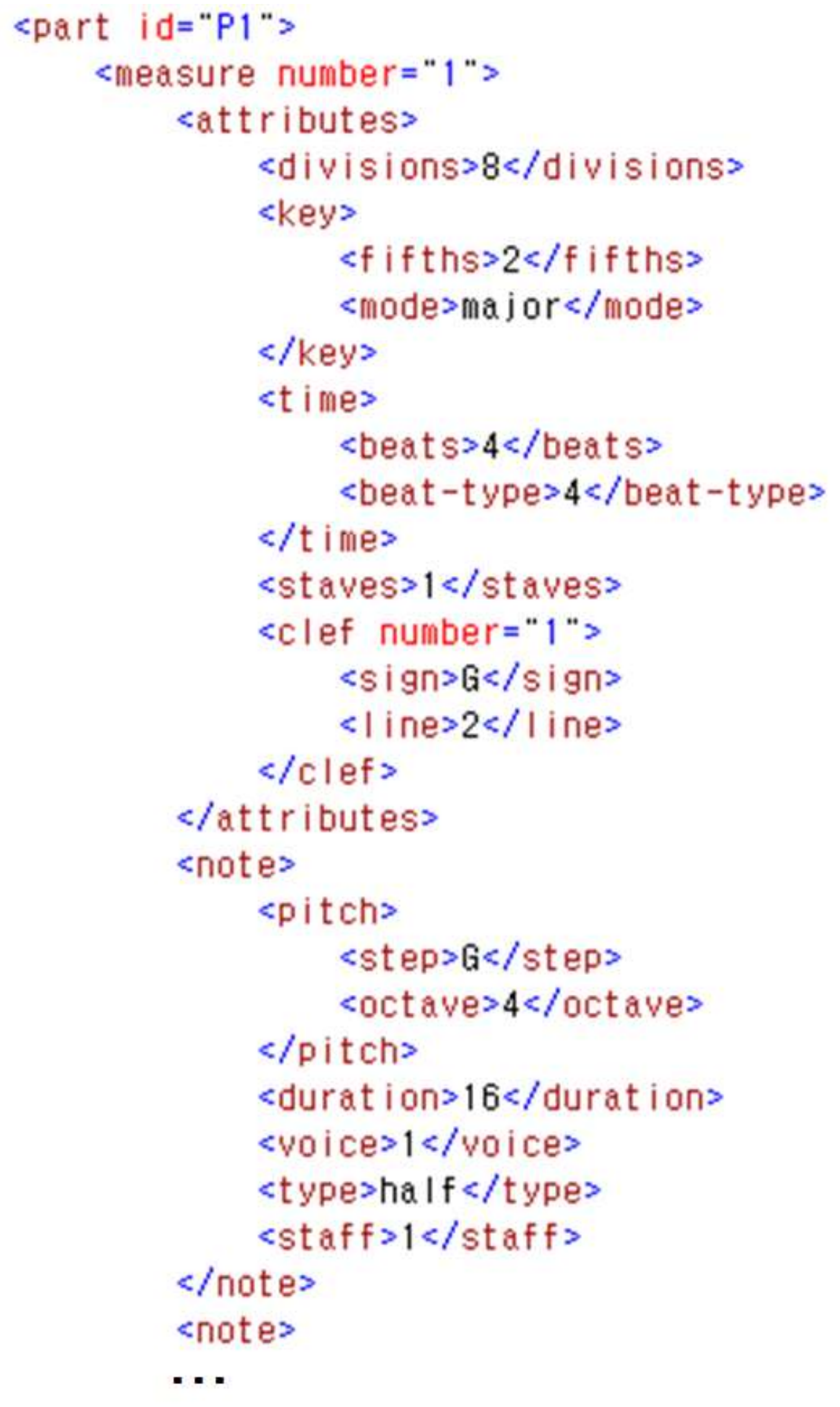


Results

Notes added to the score



Result



```
<part id="P1">
    <measure number="1">
        <attributes>
            <divisions>8</divisions>
            <key>
                <fifths>2</fifths>
                <mode>major</mode>
            </key>
            <time>
                <beats>4</beats>
                <beat-type>4</beat-type>
            </time>
            <staves>1</staves>
            <clef number="1">
                <sign>G</sign>
                <line>2</line>
            </clef>
        </attributes>
        <note>
            <pitch>
                <step>G</step>
                <octave>4</octave>
            </pitch>
            <duration>16</duration>
            <voice>1</voice>
            <type>half</type>
            <staff>1</staff>
        </note>
        <note>
        ...
```

References

- N.U. Khan and J.C. Lee "Development of a Music Score Editor based on MusicXML," Journal of The Korean Society of Computer and Information, vol.19, no.2, p77-90, Feb. 2014
- www.musicxml.org
- http://www.w3schools.com/xml/

Thank You