SeeScore Development for Android
Hints and tips

The developer will need to understand the Java language

The SeeScore API is all within the package **uk.co.dolphin_com.sscore,** and sub-packages and is very easy to use, especially if you use **SeeScoreView** and **Player** in package **uk.co.dolphin_com.seescoreandroid** described here.

uk.co.dolphin_com.sscore.**SScore**

SScore is the main class of the SeeScore API. You create an object of this class from a MusicXML file using method **loadXMLFile(),** or from MusicXML UTF-8 data in a buffer using **loadXMLData()**

There are many accessor methods in **SScore** which return useful information such as **numParts(), numBars()** and **getHeader()** and you can find information about individual items in the score (notes, rests, directions etc.) (with contents licence) using **getBarContents()** and **getXmlForItem()**, and more detailed information (with contents-detail licence) using **getXMLForBar()** and **getItemForHandle().**

There is also support for transpose using method **setTranspose(). MainActivity** shows how to do the transpose in a background thread so you don't stall the foreground thread.

uk.co.dolphin_com.sscore.**Version**

It is a good idea to display the result of **SSScore.getVersion().toString()** somewhere in the user interface of your app. This way if there is a problem you can report back to us the version of the SeeScoreLib being used.

uk.co.dolphin_com.sscore.**PlayData**

This contains all the information required to play the score. You construct the **PlayData** with the **SScore** and **UserTempo** objects. It implements **Iterable** so you can use it in a **for** loop to iterate through all the bars in the score accounting for any repeats. Within each bar you can iterate through parts and notes, and there is a special metronome part which has a single note for each beat. You can also create a count-in bar for a count-in before the first bar to play. There is also the method **PlayData.createMIDIFile()** to create a normal MIDI file from the play data. **PlayData.createMIDIFileWithControls()** allows more control of individual parts in the MIDI file. Finally there is a **PlayData** constructor which allows you to define a set of bars to repeat, to play a loop

uk.co.dolphin_com.seescoreandroid.**LicenceKeyInstance**

This contains your key which enables specific features. The downloaded SDK comes with an evaluation key which enables all features in time-limited evaluation mode, and it draws a red 'watermark' over the score. This watermark is removed when you buy a key for the features which you require

uk.co.dolphin_com.seescoreandroid.**SeeScoreView**

The simplest way to display a score in your app is to construct class **SeeScoreView** and call method **setScore()** with the **SScore** object. This immediately gives you a scrollable view of the score with an overlaid cursor. There is also support for displaying graphics for a looped play section using **displayLoopGraphics().**

uk.co.dolphin_com.seescoreandroid.**Player**

To play the score you will need a play-data licence (a time-limited licence is included with the free evaluation key). Construct the **Player** using the **SScore** object passing an object implementing the **UserTempo** interface and an object implementing the **PlayControls** interface. These two interfaces normally connect to the user interface (UI) to allow the user to control the tempo and part volumes. Note that when the user changes the tempo while playing the method **updateTempo()** must be called. This restarts playing at the start of the current bar with the new tempo. Because of pipelining considerations it is not possible to change tempo seamlessly while playing. There is a constructor which allows you to setup a repeat loop. Use **startAt()** to start playing the score on the built-in Android MediaPlayer. There are methods to pause, resume, reset. You can also setup handlers to be notified of events - bar start, beat, note start and stop and play end. You specify a delay to be used when calling the handler. The delay can be positive or negative, for example to start to move an animated cursor before the event allowing it to arrive in position exactly on the event time

uk.co.dolphin_com.seescoreandroid.**MainActivity**

shows how to put all the pieces together to make an app. It uses the **Player** to synchronise the playing of a MIDI file in the **MediaPlayer** with movement of the cursor.