Music Notation Library - Software Development Kit
Specifications - Level I

Version 1.2.0 - 28 March 2013

Version history

Version	Date	What's new
1.0.0	25 May 2012	First public release of the Music Notation Library SDK
1.1.0	7 November 2012	Import musicXML 3.0 files as well as compressed musicXML files (extension .mxl)
1.2.0	28 March 2013	The SDK is now available for iOS
1.2.1	23 April 2013	Adds the possibility to play the score as graphic animation but without real playback
1.2.2	20 June 2013	Adds the possibility to expand a score with the repeated measures.

Purpose

To provide a software development library that delivers services for music notation displaying and playing - as found in the Pizzicato music software brand of products – so that software developers may use these features without the need to develop a full music notation library by themselves.

This SDK has two levels for licenses. Level I mainly handles existing music documents for display, playback and transpose. With Level II you can also edit the music content of documents and you can create new documents with customized content.

This document describes the software library interface, provided as a C header file with a DLL library on Windows, a "dylib" dynamic library for Mac OS X and a static library for iOS.

All functions return an error code (0 if no error). The various error codes returned are defined in the MNL.h header file as well as at the end of this document. MNL means *Music Notation Library*.

Two units are used in the functions of the SDK : millimeters and pixels. The conversion from one to the other depends upon the scaling factor (by default 1.0) applied in the page layout and display. The formulas are :

*millimeters = (127.0 * scale * pixels) / 360.0;*
*pixels = (360.0 * millimeters)/(127.0 * scale);*

For any question or suggestion you may have about using this SDK, please contact :

ARPEGE MUSIC
Mr. Dominique Vandenneucker
29, Rue de l'Enseignement
4800 VERVIERS
BELGIUM
Tel: ++32-87.55.23.60
Fax: ++32-87.26.80.10
info@arpegemusic.com
www.arpegemusic.com

Creating a project

Windows

To use MNL on Windows, you must add the files *MNL.h* and *MNL.c* in your project. This will compile the code to load the DLL and give you the prototypes of the functions.

To see the music symbols properly with your application, you must install the *Pizzicato.ttf* TrueType font in Windows. Do not forget to specify this in the installer program for your application, or the user will see big letters instead of musical symbols.

You must also add the *MSL_All-DLL90_x86.dll* file in the same directory than your program, as this DLL will be loaded automatically when you load the MNL library.

You will find an example project in the folder *Example I - Visual C++ 2010 Express*. It is ready to run with Microsoft Visual C++ 2010 Express edition. You must however specify your license validation code while calling the *MNLLoad* function. If you compile the project, the compiler will attract your attention on this line automatically, as this parameter is missing.

Mac OS X

To use MNL on Mac, you must add the files MNL.h and MNL.mm in your project.

A simple example is also available for Mac, including the source files, resources and xcode project file, in the *Example* folder. The Mac example application is much similar to the Windows application.

IOS for iPad

To use MNL for iOS, you must add the files MNL.h and the library *libLibrary-I (or II)* in your project. This library must be used for the final release, so as to run on an actual device. There is also a version of this library for the iPad simulator, that you must use in debug mode when running your application on the simulator.

On iOS, you must add the MNL folder in the bundle of the application and must keep it as a folder reference (otherwise xCode will put all the files of the folder in the bundle, removing the directory structure). You must also include the *Pizzicato.ttf* file in the bundle, and specify this file in the *info-plist* file as *Fonts provided by application*. In this way, the music font will be automatically available to your application.

A simple example is also available for iPad simulator, including the source files, resources and xcode project file, in the *Example* folder. The iPad example application is much similar to the Windows and Mac applications.

The following sections will describe in details each of the functions of the MNL library.

Section I-A - Loading the library

- *long* ***MNLLoad****(char *path,char *validationCode)*

  Initialize the library and all needed resources. Only after a successful call to this function may the other functions be used.

  path is the full path (with no trailing "\" or "/") of the directory where the DLL/dylib and the resource files are located. These resource files are distributed with the SDK and must be present so that the DLL/dylib may access them. This directory must contain the following files and sub-directories:

  MusicNotationLibrary.dll
  on Mac : MusicNotationLibrary.dylib
  on iPad : nothing, as the library is linked in the application
  Data \ Chords.def
  Data \ Mst-01.tbl

Data \ Bitmaps \ Texture-5.bmp
Data \ 002.wav
Data \ ChordsDiagrams.dat
DataEN \ MusicNotationLibrary.res
DataEN \ Tool.def
DataEN \ Tool.pal
DataEN \ SYN \ GM-1.SYN
DataEN \ Tablatures.dat
DataEN \ scales.def

You will find a copy of this file structure in the folder entitled *MNLFolder*.

Note that on Windows, the directory separator is "\" and on Mac/iPad it is "/".

path can be a relative path from the current program. However, if you use a relative path and if your program uses the Windows directory functions (for instance to open or save a file with the standard file dialog box), the current working path may change. If you then try to open a file by giving a relative path, the opening may fail because of the wrong working directory.

On iPad, the path can simply be the name of the folder that is inside the application bundle, by default “MNL Folder”.

validationCode is a string with the access code you have received when buying the SDK. If the validation code is not correct, you will not be able to open or create documents.

Possible error codes are :

ERCODE_COULD_NOT_OPEN_RESOURCE_FILE
ERCODE_COULD_NOT_OPEN_CHORD_LIBRARY
ERCODE_COULD_NOT_OPEN_TABLE_FONT
ERCODE_COULD_NOT_LOCATE_MNLFOLDER
ERCODE_NONE

- *long* ***MNLRelease****(void)*

Release all resources needed by the library. Must be called only if ***MNLLoad*** was successfully called. After calling ***MNLRelease***, no other function may be called anymore.

Possible error codes are :

ERCODE_DLL_IS_NOT_LOADED
ERCODE_NONE

Section I-B - Loading and saving documents

- *long* ***MNLOpenDocument****(char *pathname,long *docID)*

Opens the document from a file specified by *pathname*. It can be a Pizzicato file (*.piz), a MIDI file (*.mid) or a musicXML file (*.xml) or a compressed musicXML file (*.mxl). The function sets the *docID* to a valid document unique identifier, that can be used in other functions to refer to an open document, but only if ***MNLOpenDocument*** returns no error.

Possible error codes are :

ERCODE_COULD_NOT_OPEN_FILE_FOR_READING
ERCODE_FORMAT_ERROR_IN_PIZ_FILE

ERCODE_COULD_NOT_IMPORT_MIDI_FILE
ERCODE_COULD_NOT_IMPORT_MUSICXML_FILE
ERCODE_INVALID_SDK_LICENCE
ERCODE_NONE

- *long* ***MNLOpenDocumentFromBuffer****(void *buffer,long bufSize,long *docID)*

  Open the document from a memory buffer pointed to by *buffer*, containing *bufSize* bytes. It can be a Pizzicato file content, a MIDI file content or a musicXML file content. The function sets the *docID* to a valid document unique identifier, that can be used in other functions to refer to an open document, but only if ***MNLOpenDocumentFromBuffer*** returns no error.

  For MIDI files and old Pizzicato files (format 3.4 or earlier), a temporary file is created in the main directory where the DLL/dylib is located. This file is removed immediately after the operation.

  Possible error codes are :

  ERCODE_IMPOSSIBLE_TO_CREATE_TEMP_FILE
  ERCODE_IMPOSSIBLE_TO_WRITE_TEMP_FILE
  ERCODE_FORMAT_ERROR_IN_PIZ_FILE
  ERCODE_COULD_NOT_IMPORT_MIDI_FILE
  ERCODE_COULD_NOT_IMPORT_MUSICXML_FILE
  ERCODE_INVALID_SDK_LICENCE
  ERCODE_NONE

- *long* ***MNLCloseDocument****(long docID)*

  Close the document specified by *docID*. The document may no longer be referenced with any of the other functions.

  Possible error codes are :

  ERCODE_INVALID_DOC_ID
  ERCODE_NONE

- *long* ***MNLSaveDocument****(long docID,char *pathname,long typeOfFile)*

  Save the document specified by *docID* to the file name and location specified by *pathname*. The *typeOfFile* parameter specifies the format (1=PIZ,2=MID,3=XML;4=PDF;5=WAV). To save a document as a PDF file, it must have a page layout defined. To save a file in audio (WAV), you must first load a sound library with a call to ***MNLOpenSoundLibrary*** and also open a MIDI port with a call to ***MNLOpenMidiPort*** with *midiOutIndex = 0.* The sound library will be used to save the score in audio (stereo, 44.1 Khz, 16 bits, uncompressed).

  Possible error codes are :

  ERCODE_COULD_NOT_SAVE_PIZ_FILE
  ERCODE_COULD_NOT_SAVE_MID_FILE
  ERCODE_COULD_NOT_SAVE_XML_FILE
  ERCODE_NO_PAGE_LAYOUT_DEFINED
  ERCODE_COULD_NOT_CREATE_PDF_FILE;
  ERCODE_INVALID_TYPE_OF_FILE
  ERCODE_NO_LIBRARY_LOADED
  ERCODE_NO_MIDI_PORT_DEFINED
  ERCODE_OUT_OF_MEMORY

ERCODE_COULD_NOT_CREATE_AUDIO_FILE
ERCODE_NONE

- *long* ***MNLSaveDocumentToBuffer****(long docID,void *buffer,long *bufSize,long typeOfFile)*

  Save the document specified by *docID* to the memory buffer *buffer*. **bufSize* must point to the actual size of the buffer. If an error occurs specifying that the buffer is too small, then the function returns the minimum buffer size in **bufSize*. You can set buffer to NULL and **bufSize* to point to a valid long and the function will write to it the size needed for the buffer. Using that method may take twice as much time, as the library must compute the file twice (especially for PDF export). A faster method is to use a large buffer (25 MB or more, according to the document size) so that it will succeed directly. In any case, always check the result of the error code. If no error occurs, **bufSize* is set to the exact size of the data. The *typeOfFile* parameter specifies the format (1=PIZ,2=MID,3=XML;4=PDF;5=WAV). To save as an audio file content (WAV), you must first load a sound library with a call to ***MNLOpenSoundLibrary*** and also open a MIDI port with a call to ***MNLOpenMidiPort*** with *midiOutIndex = 0.* The sound library will be used to save the score in audio (stereo, 44.1 Khz, 16 bits, uncompressed).

  Possible error codes are :

  ERCODE_INVALID_DOC_ID
  ERCODE_BUFFER_TOO_SMALL
  ERCODE_COULD_NOT_SAVE_PIZ_FILE
  ERCODE_COULD_NOT_SAVE_MID_FILE
  ERCODE_IMPOSSIBLE_TO_OPEN_TEMP_FILE
  ERCODE_COULD_NOT_SAVE_XML_FILE
  ERCODE_NO_PAGE_LAYOUT_DEFINED
  ERCODE_COULD_NOT_CREATE_PDF_FILE;
  ERCODE_INVALID_TYPE_OF_FILE
  ERCODE_NO_LIBRARY_LOADED
  ERCODE_NO_MIDI_PORT_DEFINED
  ERCODE_OUT_OF_MEMORY
  ERCODE_NONE

Section I-C - Getting information about a document

- *long* ***MNLGetScoreInformation****(long docID,long *totalMeasures,long *totalStaves, long *totalPages, char **title)*

  Returns the number of measures, staves and pages in the score contained in the document referenced by *docID*. If no page layout is defined in the document, the number of pages returned will be zero. This is the case for instance of a MIDI file. The *title* argument must point to a "char pointer" that will receive the address of an existing null terminated string with the title of the score. You can use this pointer but you should never modify or free that pointer, as it is related to the open musical document. This pointer is no more valid when you close the document.

  Possible error codes are :

  ERCODE_INVALID_DOC_ID
  ERCODE_NONE

- *long* ***MNLGetPageSize****(long docID,long pageNumber,long *width,long *height)*

  Returns in **width* and **height*, the size of the page *pageNumber*, from the document

referenced by *docID*. The size is given in screen pixels, with a scaling factor of 1.0. If no page layout is defined in the score, the function will return an error code.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_PAGE_NUMBER
ERCODE_NONE

- *long* ***MNLGetPageMeasures****(long docID,long pageNumber,*
*long *firstMeasure,long *lastMeasure)*

Returns in **firstMeasure* and **lastMeasure*, the first and last measure numbers inside page *pageNumber*, from the document referenced by *docID*. This may be used to know which measures are displayed in a given page.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_PAGE_NUMBER
ERCODE_NO_MEASURES_IN_PAGE
ERCODE_NONE

- *long* ***MNLGetKeySignature****(long docID,long index,long *ksig,long *measure)*

Returns the number of sharps (positive number) or flats (negative number) in the **ksig* parameter (0 for the key of C). *docID* is the document reference number. *Index* goes from 1 to the number of key signatures present in the score (in case the key changes in the middle of the score). **measure* returns the measure number in which the *index* key signature was found. If *index* is greater than the number of key signatures present, the function returns an error code. Only the key signature changes are reported by this function. In the case of a page where there is a key signature at the beginning of each system (as it is the case in standard music notation), these repeated key signatures are not reported by the function.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_KEYSIG_INDEX
ERCODE_NONE

- *long* ***MNLGetTimeSignature****(long docID,long index,long *num,long *denom,long *measure)*

Returns **num* (the numerator) and **denom* (the denominator) of the time signature (for instance 6 and 8 for a 6/8 time signature). *docID* is the document reference number. *Index* goes from 1 to the number of time signatures present in the score (in case the time signature changes in the middle of the score). **measure* returns the measure number in which the *index* time signature was found. If *index* is greater than the number of time signatures present, the function returns an error code.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_TIMESIG_INDEX
ERCODE_NONE

Section I-D - Displaying the score

- *long* ***MNLDisplayPage****(long docID,long pageNumber,long offsetX,long offsetY,float scale, long x1,long y1,long x2,long y2,long graphic)*

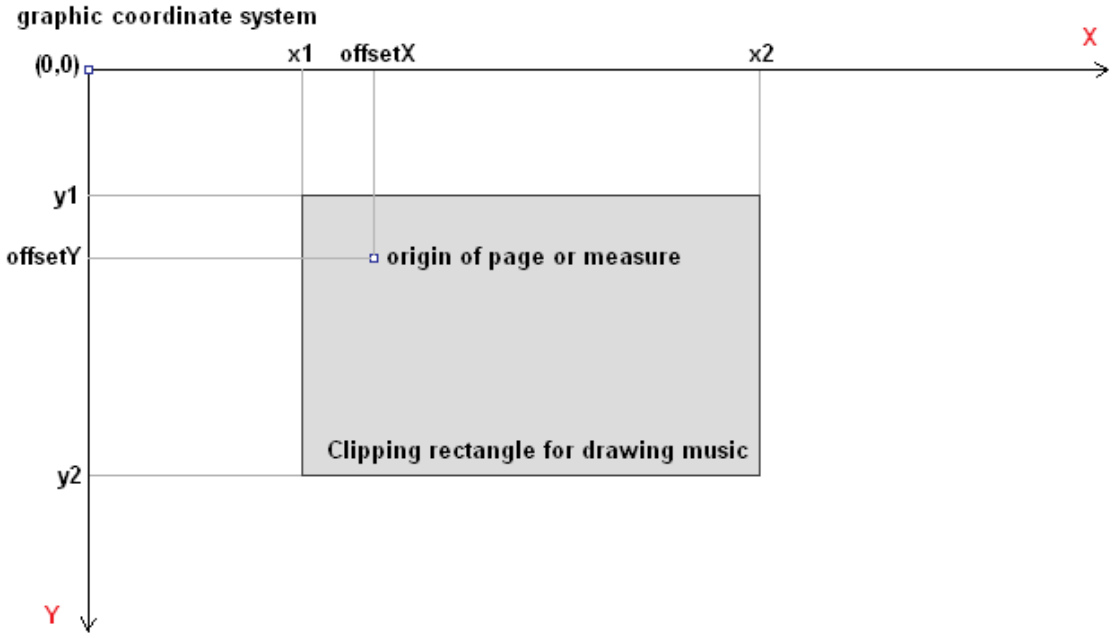
  This function displays the page *pageNumber* of the score in page mode. *docID* is the document reference number.

  For Windows, drawing is performed to the device context supplied by the *graphic* parameter. The *graphic* parameter is in fact a Windows *HDC* type (handle to a device context). The most obvious way to use this function is while handling the *WM_PAINT* Windows message:

```
LRESULT  WndProc(HWND hWnd,UINT messg,WPARAM wParam,LPARAM lParam)
{    HDC          hdc;
     PAINTSTRUCT  pstruct;

     switch(messg)
     {    case WM_PAINT:
               hdc = BeginPaint(hWnd,&pstruct);
               ercode = MNLDisplayPage(docID,1,0,0,1.0,0,0,500,800,(long) hdc);
               EndPaint(hWnd, &pstruct);
               break;
```

  The (*offsetX*, *offsetY*) parameters define the position where the top left corner of the page will be displayed. The (*x1,y1,x2,y2*) rectangle is used as a clipping rectangle for drawing. This means that anything outside this rectangle will not be drawn on the screen. *scale* is used to scale the display, 1.0 being the reference size. You can get the page size information by using ***MNLGetPageSize***, so as to determine the scale factor to use to fit the screen display area where you want the score to appear. You can use the offset parameters to determine which part of the page to display if the area is too small to display the full page. The offset parameters could for instance be linked to scroll bars, so that the user can navigate in the page. Here is how *offsetX*, *offsetY, x1,y1,x2,y2* are related to the coordinate system of *graphic*:



  For Mac, the *graphic* parameter is in fact an *NSView* object or a sub-class of it. For iOS, it must be an *UIView*. The most obvious place to use this function is in the implementation function of :

```
- (void) drawRect:(NSRect)dirtyRect
```

of the NSView/UIView object.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_PAGE_NUMBER
ERCODE_NONE

- *long* ***MNLDisplayLinear****(long docID,long firstMeasure,long totalMeasures,long offsetX, long offsetY,float scale,long x1,long y1,long x2,long y2,long graphic)*

  This function displays *totalMeasures* measures of the score, starting at measure *firstMeasure* in linear mode. *docID* is the document reference number. The linear mode displays all measures from left to right, with no attention to page layout. If *totalMeasures* is set to zero, the function will automatically compute the number of measures that can be displayed in the clipping rectangle, taking into account the scaling factor.

  The use of the *graphic* parameter is the same as for ***MNLDisplayPage***.

  The (*offsetX*, *offsetY*) parameters define the position where the top left corner of the first measure will be displayed. The (*x1,y1,x2,y2*) rectangle is used as a clipping rectangle for drawing. This means that anything outside this rectangle will not be drawn on the screen. *scale* is used to scale the display, 1.0 being the reference size.

  When displaying with *scale* = 1.0, a standard 5-lines staff is exactly 24 pixels high (6 pixels between each line).

  Possible error codes are :

  ERCODE_INVALID_DOC_ID
  ERCODE_INVALID_RANGE_OF_MEASURES
  ERCODE_NONE

- *long* ***MNLDisplayWrap****(long docID,long firstMeasure,long totalMeasures,long offsetX, long offsetY,float scale,long x1,long y1,long x2,long y2,long graphic, float minMeasScale,float maxMeasScale,long padding)*

  This function displays *totalMeasures* measures of the score, starting at measure *firstMeasure* in wrapping mode. *docID* is the document reference number. The wrapping mode displays all measures from left to right, then go to the next line, fill the line, and so on. The size of the clipping rectangle as well as the scaling factor determine how much measures are displayed. If *totalMeasures* is set to zero, the score is displayed up to the last measure or to the maximum number of measures that can be displayed on the page.

  minMeasScale and *maxMeasScale* are the minimal and maximal scale factors that can be applied to the measures in order to calculate the optimum number of measures per system. This means that the function may scale the measures horizontally but not outside these values. *minMeasScale* should be less than 1.0 and *maxMeasScale* should be greater than 1.0.

  padding is a number of pixels that are inserted vertically between systems.

  The next function (***MNLSimulateDisplayWrap***) may be used to estimate how this function will display the measures, for instance to know exactly how much measures will be displayed and what the exact height each system has (so as to compute a value of *padding* to display the systems equally on the page height. See the example source file to see how it can be used.

The use of the *graphic* parameter is the same as for ***MNLDisplayPage***.

The (*offsetX*, *offsetY*) parameters define the position where the top left corner of the first measure will be displayed. The (*x1,y1,x2,y2*) rectangle is used as a clipping rectangle for drawing. This means that anything outside this rectangle will not be drawn on the screen. *scale* is used to scale the display, 1.0 being the reference size.

Note for Windows users - When the user resizes the screen and you call ***MNLDisplayWrap*** in real time to display the score, using the standard displaying technique will create a visual flickering, as the area is first erased by Windows, then painted. This is also the case when displaying in page or linear mode. To avoid the flickering, you can draw the content of the window in an off-screen memory device context, then copy it to the screen area, without erasing the screen area first. This method is illustrated in the example provided with the SDK. The way to implement this is :

1. Before the main loop, create a device context and a bitmap object that will be used to draw the score and any other element you have in that window (keyboard, fret board, custom items of your application).
2. When registering the window class, set the *wc.hbrBackground* field to NULL, so that Windows will not erase the background before sending the *WM_PAINT* message to your window, as this erasing is the cause of the visual flickering.
3. When responding to the *WM_PAINT* message of Windows, draw everything in the off-screen device context and use the *BitBlt* Windows function to copy the image to the screen.

On Mac and iPad, as there is a natural double-buffered display mechanism in *Quartz*, there is no need to use a temporary off-screen display.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_RANGE_OF_MEASURES
ERCODE_NONE

- *long* ***MNLSimulateDisplayWrap****(long docID,long firstMeasure,long offsetX, long offsetY, float scale,long x1,long y1,long x2,long y2,float minMeasScale, float maxMeasScale,long *totalSystems,long *totalMeasuresTable, long *systemHeightTable)*

This function computes how the function ***MNLDisplayWrap*** will display the measures, starting at measure *firstMeasure* in wrapping mode. *docID* is the document reference number. The wrapping mode displays all measures from left to right, then go to the next line, fill the line, and so on. The size of the clipping rectangle as well as the scaling factor determine how much measures are displayed.

The (*offsetX*, *offsetY*) parameters define the position where the top left corner of the first measure will be displayed. The (*x1,y1,x2,y2*) rectangle is used as a clipping rectangle for drawing. This means that anything outside this rectangle will not be drawn on the screen. *scale* is used to scale the display, 1.0 being the reference size. *minMeasScale* and *maxMeasScale* are the minimal and maximal scale factors that can be applied to the measures in order to calculate the optimum number of measures per system.

totalSystems points to a long that will be set to the number of systems that can fit the rectangle. *totalMeasuresTable* is a table of long that will be set to the number of measures contained in each system. The first value is index number 1, so that the number of measures in the first system will be set in *totalMeasuresTable[1]*.

systemHeightTable is a table of long that will be set to the vertical size of the system, in pixels, including the scale factor. The first value is index number 1, so that the height of the first system will be set in *systemHeightTable[1]*.

When calling this function, you must set *totalSystems* to the maximum possible size (+1) of the tables *totalMeasuresTable[ ] and systemHeightTable[ ]*. In case that the number of systems is greater, the function will return the error value ERCODE_BUFFER_TOO_SMALL.

By adding the values of *systemHeightTable[ ]*, you know exactly how much pixels the systems will need for display. You can then calculate a *padding* value to be used to display the systems so that they are equally spaced vertically inside the rectangle, as shown in the example file.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_RANGE_OF_MEASURES
ERCODE_BUFFER_TOO_SMALL
ERCODE_NONE

- *long* ***MNLFindPosition****(long docID,long x,long y,long *measureNumber,long *beat, long *staffNumber)*

This function will locate the *measureNumber*, the *staffNumber* and the *beat* number (from 1 to the total number of beats in the measure) from the graphic position *x,y* in the display area. The display area parameters (scale, offset and rectangle) are taken from the last call to one of the display functions. *docID* is the document reference number.

The detected positions are either inside one of the measures (between two bar lines and inside the staff lines) or between two staves of the same system. In any other case, the function returns ERCODE_POSITION_NOT_INSIDE_MEASURE.

On Windows and iPad, the origin of the coordinate system is the upper left corner. On Mac, an *NSView* has by default its origin in the bottom left corner. The SDK takes that into account for all drawing operations, but as you catch a mouse event on the Mac, you get the mouse position according to the bottom left corner, so you must convert it before passing it to this function.

In the SDK example, this function is used so that the user can click on one measure - while the score is playing - to start playing that measure.

Possible error codes are :

ERCODE_NO_SCORE_DISPLAYED
ERCODE_POSITION_NOT_INSIDE_MEASURE
ERCODE_NONE

- *long* ***MNLGetMeasurePosition****(long docID,long measureNumber,long staffNumber, long *posX,long *posY)*

This function returns the position (in pixels) of the top left corner of measure *measureNumber,* on staff *staffNumber* for a scaling factor of 100 %. It is the position where the measure is actually displayed.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_MEASURE_NUMBER
ERCODE_INVALID_STAFF_NUMBER
ERCODE_NONE

- *long* ***MNLMeasureUnitsToPosition****(long docID,long measureNumber,long staffNumber, long units,long *posX)*

This function converts the time position (in *units*, one quarter note being 480 units) into the corresponding *posX* position. Both *units* and *posX* are relative to the beginning of the measure defined by *measureNumber, staffNumber* and *docID*.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_MEASURE_NUMBER
ERCODE_INVALID_STAFF_NUMBER
ERCODE_INVALID_PARAMETER_VALUE
ERCODE_NONE

- *long* ***MNLMeasurePositionToUnits****(long docID,long measureNumber,long staffNumber, long posX, long *units)*

This function converts the graphic position *posX* (in pixels) into the corresponding time position in **units*. Both *units* and *posX* are relative to the beginning of the measure defined by *measureNumber, staffNumber* and *docID*.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_MEASURE_NUMBER
ERCODE_INVALID_STAFF_NUMBER
ERCODE_INVALID_PARAMETER_VALUE
ERCODE_NONE

Section I-E - Modifying the document

- *long* ***MNLTranspose****(long docID,long ksig,long transposeUp, long fromMeasure,long toMeasure)*

Transpose the score to the *ksig* key signature (from -7, meaning 7 flats, to +7, meaning 7 sharps), from measure *fromMeasure* to measure *toMeasure*. *docID* is the document reference number. If *transposeUp* is *true*, the transposition is done upwards, otherwise it is downwards. As an example, a melody in C starting on C3, transposed to G (1 #) will start on G3 if *transposeUp* is *true* or on G2 if *transposeUp* is *false*.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_RANGE_OF_MEASURES
ERCODE_NONE

- *long* ***MNLTransposeByInterval****(long docID,long noteInterval,long midiInterval, long adaptKeySignature,long fromMeasure,long toMeasure)*

Transpose the score using the given interval specification, from measure *fromMeasure* to measure *toMeasure*. *docID* is the document reference number. *noteInterval* represents the interval in terms of note difference (0 = unison, 1 = second, 2 = third,...); it is the number of steps to add or subtract from the original note. C to D is a second, it is one note difference in the natural scale. *midiInterval* is the number of semitones in the interval. For instance, a Major third is 4 semitones. These two numbers (*noteInterval, midiInterval)* thus represent any given interval unambiguously. A perfect fourth up is then (*noteInterval = +3, midiInterval = +5*). A minor sixth down is (*noteInterval = -5, midiInterval = -8*).

The pairs (*noteInterval, midiInterval*) are not all possible. Only valid standard intervals are accepted by the function: unison, second, third, fourth, fifth, sixth, seventh or their octaves, and they must be perfect, augmented, diminished, minor or major.

As a special case, if *midiInterval* is greater than or equal to 1000 (which is no valid midi interval anyway) then the transposition is done simply by shifting the notes on the staff, with no modification of the accidentals. This is used for instance to transpose between different clefs. With a value of 1000, the transposition is done on all staves. With value 1001, it is done only on staff 1, with 1002 on staff 2,...

If *adaptKeySignature* is TRUE, then the key signature is adapted accordingly to the resulting transposition. Depending on the accidentals present in the measures, and if the resulting key signature is outside the range of [7 flats, 7 sharps], the result is not guaranteed, as some notes may become impossible to specify (for instance notes that would require a triple sharp or flat).

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_RANGE_OF_MEASURES
ERCODE_INVALID_INTERVAL_SPECIFICATION
ERCODE_INVALID_PARAMETER_VALUE
ERCODE_NONE

- *long* ***MNLNewPageLayout****(long docID,long paperWidth,long paperHeight,float scale,*
  long leftMargin,long rightMargin,
  long topMargin, long bottomMargin
  unsigned long staffFlags,long staffChords)

This function will create a new page layout for the score. *docID* is the document reference number. *paperWidth* and *paperHeight* represent the size of the paper, in millimeters. *scale* can be used to increase the global size of the music (1.0 is standard 100 % display). *leftMargin, rightMargin, topMargin* and *bottomMargin* specify the margins in millimeters. If set to zero, the music will fill the whole page.

The *staffFlags* parameter is used to select which staves are displayed. If set to zero, then all staves are displayed. Otherwise, bit 0 must be set to display staff 1, bit 1 must be set to display staff 2, and so on. At least one existing staff must be visible, or the function will display automatically staff 1. The visible staves will also be used while displaying the score in wrapping mode.

staffChords is used to specify on which staff to display the chord symbols. If set to -1, it will not affect the existing chord symbols. If set to 0, no chord will be displayed. Otherwise, the chord symbols will be displayed above staff number *staffChords*.

This function must be called for an imported MIDI file (as a MIDI file does not have a page layout defined) before you can print it to a PDF file or display it in page view. Please note

that some musicXML or Pizzicato file may also not have a default page layout when loaded in memory. You can also use this function to change the page layout of the document.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_NONE

- *long* ***MNLExpandRepeats****(long docID,long *newDocID,long options)*

  This function will create a new document with the original docID score but that contains all the repeated measures one after the others. The new document will have the *newDocID* if it is successful. By playing the new score, you will have the same result as playing the original score, but as the repeated passages are duplicated, there is no need to jump to another part of the score whenever a repeat happens.

  If bit 0 of *options* is set, the function will reset the measure numbers, otherwise it will keep the original measure numbers from the score.

  Possible error codes are :

  ERCODE_INVALID_DOC_ID
  ERCODE_COULD_NOT_CREATE_DOCUMENT
  ERCODE_NONE

Section I-F - Playing the score

- *long* ***MNLOpenMidiPort****(long midiOutIndex)*

  This function will open and prepare a MIDI output port to play a score. It must be called successfully before starting to play a score. This call must be matched with a call to ***MNLCloseMidiPort*** to close the MIDI port. Between these two calls, the Midi port is open and the internal Midi manager is running and using a one millisecond timer callback, for handling the timing of a Midi output flow. This timer also refreshes the graphic pointer following the playback, the note coloring, as well as the piano keyboard and guitar fret board.

  To play a score on a standard MIDI output of the computer, *midiOutIndex* must be a valid index between one and the total number of MIDI ports available.

  To play the score with a sound library, set *midiOutIndex* to zero, but you must first load a sound library by calling ***MNLOpenSoundLibrary***.

  By using a value of -1, you can create an inactive MIDI output so that you can start playing the score with graphic animation only and without real playback. When you have finished playing, you must however close this inactive MIDI port with a call to ***MNLCloseMidiPort,*** because the SDK has allocated and prepared the playback data structures.

  On Windows, the total number of MIDI outputs is returned by *midiOutGetNumDevs( )*.

  On Mac, use *midiOutIndex* = 1 to use the standard Quicktime instrument library. Values 2 and up are used to access possibly defined MIDI port on Mac.

  On iPad, there is no General Midi synthesizer by default and no MIDI port. To play the score, you can either use an external MIDI connection (you need a physical synthesizer to connect to it), or you can use a sound library (see further) or you can design your own synthesizer by setting up a *AUSampler* in iOS (this should create a MIDI port, so that you can use the MIDI

port to play the score).

Possible error codes are :

ERCODE_MIDI_PORT_ALREADY_OPEN
ERCODE_INVALID_MIDI_PORT_INDEX
ERCODE_NO_LIBRARY_LOADED
ERCODE_NONE

- *long* ***MNLCloseMidiPort****(void)*

  This function will close the MIDI output port that was open and prepared by ***MNLOpenMidiPort***.

  Possible error codes are :

  ERCODE_NO_MIDI_PORT_DEFINED
  ERCODE_STILL_PLAYING
  ERCODE_NONE

- *long* ***MNLOpenSoundLibrary****(char *libraryName)*

  This function opens a sound library of audio samples, so as to play the score without using the MIDI standard interface of Windows or Mac. The library must be prepared with Pizzicato Professional 3.6, from any General Midi SoundFont file library (file extension ".sf2"). This kind of library may be loaded freely or bought on the Internet, according to the use for commercial purposes or not. You are responsible for respecting the copyrights and license rights of the library you want to use.

  libraryName is the name of the prepared library. The MNL can use SoundFont files to play the music, but a format conversion must be executed first, with Pizzicato Professional 3.6 (a downloadable license of Pizzicato Professional 3.6 is included in the price of the SDK). Here is how to do it.

  Let us say that the SoundFont file you want to use is called "*mysounds.sf2*". It must be a General Midi compatible sound file, with 128 instruments + drum kit, otherwise the instruments may be wrong while playing (supposing that the music files you load also use General Midi).

  - Start Pizzicato 3.6 Professional
  - In the document manager (left part of the screen in Pizzicato), right-click on the folder "My scores" and select *Import a soundfont file...* and select your file *mysounds.sf2*.
  - In the next dialog box, keep the options by default (*Editable sampler* and *Sort by banks and instruments*) and click on *Import*.
  - When the operation is done, exit Pizzicato.
  - You must now copy the file and samples to the path of the MNL library, so that the program will be able to load all sound data. Copy the file named *mysounds.piz* from "*my documents / Pizzicato 3.x / My scores*" to the main path of the library (as defined by a call to ***MNLLoad***). You must also copy the folder *mysounds* from "*my documents / Pizzicato 3.x / DataEN / Libraries / Music libraries / Audio / SoundFonts*", to the main path of the library. This folder contains all the sample files of individual instruments and notes. Do not forget to add these files in your program

installer, so that the end user can use them.

When this is done, you can call ***MNLOpenSoundLibrary****("mysounds")* to load the library for playback.

On iOS, the folder and the file must be placed in the application bundle.

You can edit the *mysounds.piz* file if you want, for instance to reorganize the sound library or to create a sound library of your own. You will find more information on that in the Pizzicato User Manual.

For more information about importing soundfont files:

http://www.arpegemusic.com/manual36/EN930.htm#J4

For information about how to edit an audio synthesizer:

http://www.arpegemusic.com/manual36/EN940.htm#J6

Pizzicato Professional uses a General Midi sound library, licensed from the Papelmedia company in Germany. You can use this library while using Pizzicato Professional 3.6, or for testing the SDK, but **you do not have the rights to include that library in your applications for distribution.** If you want to use that library in your applications, please contact us and we will set you in contact with the owner of the rights of that library.

Once a sound library has been loaded successfully, you can call ***MNLOpenMidiPort*** with the *midiOutIndex* set to zero, to prepare playback to use this sound library. You can also save a document as an audio wave file (file extension ".wav") using the sound from the library.

If a MIDI port is already open, you must first close it with ***MNLCloseMidiPort***.

The correct sequence is to load the sound library and then open a MIDI port with a value of zero. The playback will then be done using the sound library.

Possible error codes are:

ERCODE_MIDI_RUNNING
ERCODE_LIBRARY_ALREADY_LOADED
ERCODE_COULD_NOT_LOAD_LIBRARY
ERCODE_NONE

- *long* ***MNLCloseSoundLibrary****(void)*

  This function releases the sound library. If you have opened a MIDI port, you must first close it with ***MNLCloseMidiPort***.

  Possible error codes are:

  ERCODE_MIDI_RUNNING
  ERCODE_NO_LIBRARY_LOADED
  ERCODE_NONE

- *long* ***MNLIdle****(void)*

  This function must be called in the loop of the main program, regularly enough so that the real time display of the notes playing and the pointer following the playback is smooth enough. In

the case of Windows, if can be done as follows :

```
SetTimer(hWnd,1,20,NULL);

// Message loop

while(GetMessage(&lpMsg,NULL,0,0))
{           TranslateMessage(&lpMsg);
            DispatchMessage(&lpMsg);
}

KillTimer(hWnd,1);
```

Then, in the message handler, add the following case :

```
case WM_TIMER:
            MNLIdle();
            return DefWindowProc(hWnd,messg,wParam,lParam);
```

On Mac and iPad, create a *NSTimer* object:

```
NSTimer     *myTimer;

// Setup timer

myTimer = [NSTimer scheduledTimeWithTimeInterval:0.05 target:self
selector:@selector(applicationIdle) userInfo:NULL repeats:TRUE];

...

// Release timer

[myTimer invalidate];
```

Then you can define the *applicationIdle* method of the main application object as :

```
- (void)applicationIdle
{           MNLIdle();
}
```

Possible error codes are :

ERCODE_NONE

<u>Important note for iOS users.</u>

For iOS, there is an important difference in the way the real time animation of the notes and cursor is done. As there is no way to directly draw on an UIView outside of its *drawRect* method, the SDK uses the same *drawRect* method (through your application) to draw the score itself as well as the animation (which are note heads and the moving cursor).

For the animation to take place correctly on iPad, you must set the background color of the view to *nil* and the clearsContextBeforeDrawing to *NO*.

To avoid redisplaying the full score with each step of the animation, and as iOS uses a double-buffered mechanism to display the content of a UIView, the two following functions help you optimizing the display. They are only implemented for iOS.

- *long* ***MNLAnimation****(long graphic, long *needToRedrawAll,long doAnimation)*

  You must call this function in the *drawRect* of your application, before displaying the score. First call it with *doAnimation=false*. The function will return in **needToRedrawAll* the boolean value saying if you need to update the full score or not.

  If the **needToRedrawAll* returns false, then you can call the function again with

doAnimation=true. This will display the animation. Then you can simply return.

Otherwise, you can first display the score (using the display functions explained above) and then call the function again with *doAnimation=true* so that the animation is displayed on top of the score itself.

Possible error codes are :

ERCODE_NONE

- *long* ***MNLSyncAnimation****(void)*

  During playback and animation of the score, the *drawRect* method of your view may be called quite frequently. The above mechanism provides a way to avoid displaying the full score each time, limiting the redraw operation by coloring only the note heads that needs update.

  If during the animation you change the display, for instance by moving the score to the next measure, by changing the zoom, when the user has changed the orientation of the screen,... it can bring a situation where the *MNLAnimation* function does not know that the main score view should really be updated and will return *needToRedrawAll=false*. In this case (when you refresh the application view because you decided it), you must call *MNLSyncAnimation* to inform the SDK of this. The symptom would otherwise be that every two animations, the view will flip two different score displays, or that the animation will appear on a black background view.

  Possible error codes are :

  ERCODE_NONE

- *long* ***MNLStartPlaying****(long docID,long fromMeasure,long toMeasure,*
  *callbackWhenFinishedProc *myCallbackWhenFinished*
  long graphic,long playbackMode)

  Starts playing the score referenced by *docID* on the current Midi port defined by ***MNLOpenMidiPort***. The score is played from measure *fromMeasure* to *toMeasure*. When playing is done or a call to ***MNLStopPlaying*** interrupts the playback, the function *myCallbackWhenFinished* is called, or nothing is done if you pass NULL to this parameter. This callback function is defined as :

  void *myCallbackWhenFinished(void);*

  The *graphic* parameter is a Windows *HDC* (handle to a device context) that must be valid to display the cursor following the playback as well as coloring the notes played. For Mac and iPad, it must be a valid *NSView/UIView* object or sub-class of it. If NULL, then no graphic animation will be displayed. The graphic parameters (zoom, clip rectangle and position of the measures) are the ones that have been used the most recently on that score (through a call to one of the three display functions). If no display function has been called for the score before starting to play, then the real time animation will not be available.

  playbackMode is used only to specify how the audio playback is processed. For a standard MIDI port, this parameter is ignored. While playing with a sound library, two modes are possible:

  - *playbackMode = 0* : The program computes the sound into memory buffers in advance of the playback. This mode is more suited to scores with a lot of staves and high density of notes, but may require a lot of memory.

- playbackMode = 1 : The program computes the sound in real time. When there is only a few staves to play, or when the general note density is not too high, this mode should work properly. For instance, piano music should play correctly in that mode.

Possible error codes are :

ERCODE_ALREADY_PLAYING
ERCODE_NO_MIDI_PORT_DEFINED
ERCODE_INVALID_DOC_ID
ERCODE_INVALID_RANGE_OF_MEASURES
ERCODE_INVALID_MODE_SPECIFICATION
ERCODE_NONE

- *long* ***MNLStopPlaying****(void)*

This function will stop the playback of the score that is now playing.

Possible error codes are :

ERCODE_NO_SCORE_PLAYING
ERCODE_NONE

- *long* ***MNLGetPlayingPosition****(long *measure,long *beat,long *unit)*

This function returns the current playing *measure* number (from 1 to the total number of measures), *beat* number (from 1 to the total number of beat in that measure) and *units* (one quarter note is 480 units). It is used to know exactly where the music is playing right now.

Possible error codes are :

ERCODE_NO_SCORE_PLAYING
ERCODE_NONE

- *long* ***MNLSetPlaybackOptions****(long docID,long symbolFlags,long midiFlags)*

The playback options are various flags telling the library which effect to play through MIDI commands. *docID* is the document reference number.

A score has 3 sources of effects :

1. Initial values: these values are used to initialize the MIDI synthesizer before starting to play.
2. MIDI commands that are associated in the MIDI tracks related to the staves.
3. MIDI commands that are interpreted from the graphic symbols like nuances, tempo markings, ...

This function enables/disables the playback of MIDI commands coming from the MIDI tracks (*midiFlags*) and the commands coming from the graphic symbols (symbolFlags), for the following effects :

- 0x0001 : MIDI volume
- 0x0002 : velocity
- 0x0004 : tempo
- 0x0008 : duration

- 0x0010 : modulation
- 0x0020 : reverb (Midi controller 91)
- 0x0040 : Pitch bend
- 0x0080 : Start of notes
- 0x0100 : chorus (Midi controller 93)
- 0x0200 : Polyphonic After Touch
- 0x0400 : Monodic After Touch
- 0x0800 : Patch (program change)
- 0x1000 : Other controllers

The values may be combined as logical OR. By default, all flags are enabled, so the default values are 0x1FFF for both flags.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_NONE

- *long* ***MNLGetPlaybackParameter****(long docID, long staffNumber,long paramNumber, long *paramValue)*

This function returns in *paramValue*, the value of the playback parameter *paramNumber*, for the staff *staffNumber*. *docID* is the document reference number. The possible values of *paramNumber* are:

- 1 : Tempo value, in quarter note per minute (from 1 to 500). For the tempo, the *staffNumber* is ignored, but should be in the valid range of available staves in the score.
- 2 : MIDI volume, from 0 to 127
- 3 : MIDI pan (left to right, 0 to 127 , 64 is central position)
- 4 : Program change (instrument), from 0 to 127, in General Midi standard.
- 5 : returns the MIDI channel (from 0 to 15) used by that staff.

The value returned is the default value that is sent before playing the score.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_STAFF_NUMBER
ERCODE_INVALID_PARAMETER_NUMBER
ERCODE_NONE

- *long* ***MNLSetPlaybackParameter****(long docID, long staffNumber,long paramNumber, long paramValue)*

This function specifies in *paramValue*, the value of the playback parameter *paramNumber*, for the staff *staffNumber*. *docID* is the document reference number. The possible values of *paramNumber* are the same than for ***MNLGetPlaybackParameter***, except for value 5, as the MIDI channel is fixed by the document.

Possible error codes are :

ERCODE_INVALID_DOC_ID
ERCODE_INVALID_STAFF_NUMBER
ERCODE_INVALID_PARAMETER_NUMBER

ERCODE_INVALID_PARAMETER_VALUE
ERCODE_NONE

- *long* ***MNLDisplayKeyboard****(long keyWidth,long x1,long y1,long x2,long y2, long octave,long graphic)*

  Defines a keyboard graphic area, where notes playing in a score will be displayed in real time.

  keyWidth is the width of a white key, in screen pixels. A value of zero will adapt the width of the keys according to the height of the keyboard.

  x1,y1,x2,y2 specify the rectangle where the keyboard will be displayed in the graphic area.

  Octave is the first octave that will be displayed on the keyboard. A value of 3 will display the first note as C3 (midi pitch value 60).

  For Windows, *graphic* is a device context handle that must stay valid until the keyboard is released. On Mac, it is an *NSView* object or sub-class of it.

  This function defines the keyboard. It can be released with ***MNLReleaseKeyboard***. It can be redrawn with ***MNLRefreshKeyboard***.

  Possible error codes are :

  ERCODE_KEYBOARD_ALREADY_DEFINED
  ERCODE_NONE

- *long* ***MNLKeyboardMidiFilter****(long midiFilter)*

  Defines the MIDI channels which will be displayed on the keyboard during playback. If bit 0 of *midiFilter* is set to 1, then notes of MIDI channel 0 will be displayed. Same for bits 1 to 15. You can find which MIDI channel is used on which staff by using the function ***MNLGetPlaybackParameter*** with *paramNumber* = 5.

  Possible error codes are :

  ERCODE_NO_KEYBOARD_DEFINED
  ERCODE_NONE

- *long* ***MNLReleaseKeyboard****(void)*

  Releases the keyboard that has been created with ***MNLDisplayKeyboard***.

  Possible error codes are :

  ERCODE_NO_KEYBOARD_DEFINED
  ERCODE_NONE

- *long* ***MNLRefreshKeyboard****(long graphic)*

  This function will redraw the keyboard by using the *graphic* parameter. For Windows, it is casted to the HDC type. On Mac, it is an *NSView* object or sub-class of it.

  The keyboard may be displayed in two different ways. The first way is when the window is refreshed by the operating system, in which case you should use the ***MNLRefreshKeyboard*** with the *graphic* parameter provided by the operating system. The other way happens during

playback, when notes are displayed as a result of playback. In that case, the keyboard is updated through the next call to ***MNLIddle*** and the drawing is done on the *graphic* parameter provided when calling ***MNLDisplayKeyboard*** to define the keyboard

- *long* ***MNLDisplayFretboard****(long fretDist,long x1,long y1,long x2,long y2,long graphic)*

  Defines a fretboard graphic area, where notes playing in a score will be displayed in real time.

  fretDist is the distance between the two first frets, in screen pixels. A value of 60 is quite standard, but according to the width of the fret board you want to display, you may adapt this value.

  x1,y1,x2,y2 specify the rectangle where the fretboard will be displayed in the graphic area.

  For Windows, *graphic* is a device context handle that must stay valid until the fretboard is released. On Mac, it is an *NSView* object or sub-class of it.

  This function defines the fretboard. It can be released with ***MNLReleaseFretboard***. It can be redrawn with ***MNLRefreshFretboard***.

  Possible error codes are :

  ERCODE_FRETBOARD_ALREADY_DEFINED
  ERCODE_NONE

- *long* ***MNLFretboardMidiFilter****(long midiFilter)*

  Defines the MIDI channels which will be displayed on the fretboard during playback. If bit 0 of *midiFilter* is set to 1, then notes of MIDI channel 0 will be displayed. Same for bits 1 to 15. You can find which MIDI channel is used on which staff by using the function ***MNLGetPlaybackParameter*** with *paramNumber* = 5.

  Possible error codes are :

  ERCODE_NO_FRETBOARD_DEFINED
  ERCODE_NONE

- *long* ***MNLRefreshFretboard****(long graphic)*

  This function will redraw the fretboard by using the *graphic* parameter. For Windows, it is casted to the HDC type. On Mac, it is an *NSView* object or sub-class of it.

  The way the fretboard is displayed is the same as explained for the keyboard.

- *long* ***MNLReleaseFretboard****(void)*

  Releases the fretboard that has been created with ***MNLDisplayFretboard***.

  Possible error codes are :

  ERCODE_NO_FRETBOARD_DEFINED
  ERCODE_NONE

- *long* ***MNLResetMidi****(long midiResetMode,long keyboardReset,long fretboardReset)*

  This function resets the MIDI port according to the following *midiResetMode* values:

0 : no reset
1 : sends NOTE OFF midi messages for all notes that have been tracked to be playing.
2 : sends an ALL NOTES OFF on the open MIDI port, for all MIDI channels
3 : sends NOTE OFF midi messages for all notes and all MIDI channels

In cases 1, 2 and 3, the sustain pedal is also switched off with a MIDI controller message.

If *keyboardReset* is non zero, all notes graphically displayed on the keyboard are cleared.

If *fretboardReset* is non zero, all notes graphically displayed on the fretboard are cleared.

Possible error codes are :

ERCODE_NO_MIDI_PORT_DEFINED
ERCODE_NO_KEYBOARD_DEFINED
ERCODE_NO_FRETBOARD_DEFINED
ERCODE_INVALID_PARAMETER_VALUE
ERCODE_NONE

****

Appendix A - Error codes

All functions return an error code. You should always handle error cases, as ignoring them may cause the program to crash.

0	ERCODE_NONE
1	ERCODE_COULD_NOT_LOAD_DLL
2	ERCODE_DLL_IS_NOT_LOADED
3	ERCODE_DLL_ALREADY_LOADED
4	ERCODE_COULD_NOT_OPEN_FILE_FOR_READING
5	ERCODE_FORMAT_ERROR_IN_PIZ_FILE
6	ERCODE_COULD_NOT_OPEN_CHORD_LIBRARY
7	ERCODE_COULD_NOT_OPEN_RESOURCE_FILE
8	ERCODE_COULD_NOT_IMPORT_MIDI_FILE
9	ERCODE_COULD_NOT_OPEN_TABLE_FONT
10	ERCODE_COULD_NOT_IMPORT_MUSICXML_FILE
11	ERCODE_INVALID_DOC_ID
12	ERCODE_INVALID_TYPE_OF_FILE
13	ERCODE_COULD_NOT_SAVE_PIZ_FILE
14	ERCODE_COULD_NOT_SAVE_MID_FILE
15	ERCODE_COULD_NOT_SAVE_XML_FILE
16	ERCODE_COULD_NOT_CREATE_PDF_FILE
17	ERCODE_NO_PAGE_LAYOUT_DEFINED
18	ERCODE_IMPOSSIBLE_TO_CREATE_TEMP_FILE
19	ERCODE_IMPOSSIBLE_TO_WRITE_TEMP_FILE
20	ERCODE_BUFFER_TOO_SMALL
21	ERCODE_IMPOSSIBLE_TO_OPEN_TEMP_FILE
22	ERCODE_INVALID_PAGE_NUMBER
23	ERCODE_INVALID_KEYSIG_INDEX
24	ERCODE_INVALID_TIMESIG_INDEX
25	ERCODE_INVALID_RANGE_OF_MEASURES
26	ERCODE_ALREADY_PLAYING
27	ERCODE_NO_SCORE_PLAYING
28	ERCODE_INVALID_MIDI_PORT_INDEX
29	ERCODE_MIDI_PORT_ALREADY_OPEN
30	ERCODE_NO_MIDI_PORT_DEFINED
31	ERCODE_STILL_PLAYING
32	ERCODE_NO_MEASURES_IN_PAGE
33	ERCODE_INVALID_STAFF_NUMBER
34	ERCODE_INVALID_PARAMETER_NUMBER
35	ERCODE_INVALID_PARAMETER_VALUE
36	ERCODE_NO_SCORE_DISPLAYED
37	ERCODE_POSITION_NOT_INSIDE_MEASURE
38	ERCODE_KEYBOARD_ALREADY_DEFINED
39	ERCODE_NO_KEYBOARD_DEFINED
40	ERCODE_FRETBOARD_ALREADY_DEFINED
41	ERCODE_NO_FRETBOARD_DEFINED
42	ERCODE_LIBRARY_ALREADY_LOADED
43	ERCODE_NO_LIBRARY_LOADED
44	ERCODE_MIDI_RUNNING

45	ERCODE_COULD_NOT_LOAD_LIBRARY
46	ERCODE_INVALID_MODE_SPECIFICATION
47	ERCODE_COULD_NOT_CREATE_AUDIO_FILE
48	ERCODE_OUT_OF_MEMORY
49	ERCODE_INVALID_INTERVAL_SPECIFICATION
50	ERCODE_COULD_NOT_CREATE_DOCUMENT
51	ERCODE_INVALID_MEASURE_NUMBER
52	ERCODE_INVALID_OBJECT_INDEX
53	ERCODE_INVALID_OBJECT
54	ERCODE_INVALID_LYRIC_LINE_NUMBER
55	ERCODE_INVALID_CHORD_INDEX
56	ERCODE_INVALID_MIDIEVENT_INDEX
57	ERCODE_INTERNAL_ERROR
58	ERCODE_INVALID_RANGE_OF_STAVES
59	ERCODE_INVALID_SYSTEM_NUMBER
60	ERCODE_INVALID_SDK_LICENCE
61	ERCODE_MNL_ALREADY_LOADED
62	ERCODE_MNL_NOT_LOADED
63	ERCODE_COULD_NOT_LOCATE_MNLFOLDER