ReadScoreLib Optical Music Recognition Library Version 2.5 November 2016

ReadScoreLib from Dolphin Computing is a comprehensive OMR system for converting printed/engraved music notation to MIDI and MusicXML. The library and output is fully compatible with Dolphin's SeeScore SDK which provides rendering and MusicXML query capabilities.

The performance of the library varies according to platform but on Windows a page of music takes about 2 seconds to translate. Music images can be processed from memory or from file, by the page or in bulk. ReadScoreLib can cope with photographically distorted images and can process pages photographed with a device camera. ReadScoreLib can compensate for the missing rests and tuplet marks which are common in printed music, and can deal with variable size systems, missing time signatures and transposing instruments. The MIDI files generated by ReadScoreLib play naturally and continuously and without the jerks and jumps sometimes seen in OMR output. ReadScoreLib is also the only OMR system to support the varied tremolo notations often found in printed music.

Recognition from a memory resident bitmap image

rscore_convert accepts a page of music in memory and generates corresponding MIDI and MusicXML files subject to selected options

rscore*	**rscore_convert(**		**read an image (32bpp RGBA) and convert to MIDI and/or XML files**
	const unsigned*	image	array of 32bit pixels in (32bpp RGBA) format with the alpha channel (bits 24 – 31) should be zero
	int	width	the pixel width of the image
	int	height	The pixel height of the image
	int	rowbytes	the byte offset from one row to the next and must be a multiple of 4
	enum rscore_imageorientation	orientation	the orientation of the image. See rscore_imageorientation
	rscore_progress_callback	cb	user-supplied callback called periodically to report progress and allow the process to be abandoned pass NULL if no callback desired
	Void*	arg	the context argument to be passed to cb
	const char*	midifilepath	if non-null the MIDI file is written to the full path midifilepath
	const char*	xmlfilepath	if non-null the MusicXML file is written to the full path xmlfilepath

	const rscore_options*	options	Information including bits specifing selected options (see rscore_optionsflags)
	Unsigned*	Image_out	for use only when rscore_optLens selected: rectilinearised image written to image_out. Set NULL otherwise
	rscore_errorinfo*	err	if non-null points to a struct which receives any error information
	Return		handle to rscore instance. handle should be deleted on completion with rscore_delete

Image dimensions

The input image is laid out as a series of concatenated rows each pixel of which occupies one 32-bit word. image points to the top left word. The layout is RGB with the remaining (ms) byte set to zero.

The width and height arguments are in pixel units. The *rowbytes* argument allows a section of a larger image to be processed. This is done by setting image to the top left word of the subimage, width and height to those of the *subimage* and *rowbytes* the width of the full image.

Music segments

ReadScoreLib can recognise multiple areas of musical notation as for example might be found on a page of text containing musical examples. The positioning and size of these areas may be varied but will be read as a vertical sequence. Multiple columns of music are not currently supported.

Where there are multiple areas, or where a page containing multiple short pieces, or where one piece ends and another begins, the output MIDI and MusicXML will be continuous (as anything else would alter the music). However, by using *rscore_getbarinfo* and in particular the *rscore_firstOfSection* flag it is possible for an application to identify boundaries. And by using coordinate information (*see rscore_optXmlDefaultCoords* and *optXmlObjectBounds*) a rich graphical application is possible.

Preprocess only mode (see *rscore_optLens*)

For some applications access may be needed to the preprocesed image (see table below). This can be useful where images are distorted or poorly defined. The preprocessed image in black and white with distortions removed is often easier for a human to read and may be more amenable to further graphical processing.

When the *rscore_optLens ptions* flag is set only preprocessing takes place with the resulting image passed back through *the image_out* argument. The user supplied memory bitmap

specified *by image_out* has exactly the same format as image except that *rowbytes* will be ignored and the image will be written to a contiguous block of memory of size (*width * height) * 4*.

NB The ReadScoreLib *rscore_optXmlObjectBounds* feature provides bounding box information for musical objects relative to the original input image; that is before preprocessing. If an application requires bounding box information relative to the preprocessed image it can be obtained by first preprocessing the image using the *rscore_optXmlObjectBounds* option and then submitting the preprocessed image for normal recognition without the flag.

NB Lens processing can only be performed using *rscore_convert*, one image at a time.

Progress callback

The *cb* argument allows the caller to specify a function to be called periodically to report progress as a percentage. *cb* should conform to the prototype

*typedef bool (*rscore_progress_callback)(const rscore_convert_progressinfo *info, void *arg);*

for example

```
bool cb(const rscore_convert_progressinfo *info, void *arg) {
        printf("%d% complete ", info->progress_percent);
        return true;
}
```

cb will be called by ReadScoreLib with the *progress_percent* member of the passed in *rscore_convert_progressinfo* struct giving the percentage of the job completed.

cb should normally return true. A false return will cause the job to be abandoned and control from *rscore_convert* to return.

Recognition from image files

rscore_fconvert accepts one or more images of music notation and generates corresponding MIDI and MusicXML files subject to selected options

rscore*	**rscore_fconvert(**		**read one or more page image files and output corresponding MIDI and MusicXML files**
	const char*	buildfilepath	single image file - the full path to the image file multiple image files – the full

			path to a build file containing a list of JPG image files
	const char*	midifilepath	if non-null the MIDI file is written to the full path midifilepath. Otherwise it is written to the build file directory
	const char*	xmlfilepath	if non-null the MusicXML file is written to the full path xmlfilepath. Otherwise it is written to the build file directory
	rscore_progress_callback	cb	user-supplied callback called periodically to report progress and allow the process to be abandoned pass NULL if no callback desired
	Void*	arg	the context argument to be passed to cb
	rscore_options*	options	Information including bits specifing selected options (see rscore_optionsflags)
	rscore_errorinfo*	err	if non-null points to a struct which receives any error information
	Return		handle to rscore instance. handle should be deleted on completion with rscore_delete

rscore_fconvert operates in two modes. If *buildfile* is an image file (has one of the supported image extensions[1]) that image will be treated as the sole input. If *buildfile* is not an image file it will be read as a text file containing the list of image files to be built together as a single unit. The named build file should consist of a list of image names, one per line and without paths. For example if mybuild.txt contains the following

myimage1.jpg
myimage2.jpg
myimage3.jpg

a call to *rscore_fconvert* giving the full path to mybuild,txt would look for myimage1.jpg, myimage2.jpg and myimage3.jpg in the same directory as mybuild.txt and treat the three as a single, piece of music. A single MIDI and a single MusicXML file will be generated.

Unlike *rscore_convert* MIDI and MusicXML files are always generated. By default, if *midifilepath* is NULL a MIDI file, named after the build file will be written to the same directory. If *midifilepath* is a full path, name and extension, the MIDI file will be written there. *xmlfilepath* is treated in exactly the same way.

[1] Currently JPG only

Progress callback

ReadScoreLib processes music in two stages Rather than treat each page as a separate unit *rscore_fconvert* creates a syntax tree for the whole input, page by page. When all files have been read the remaining steps are applied to the resulting internal representation as a single unit. This lets ReadScoreLib to carry out time and key analysis on the whole piece as well as allowing bars, arcs and other structures which span a page boundary to be handled correctly.

Because of this organisation the progress callback functions for *rscore_convert* and *rscore_fconvert* work differently. Whereas *rscore_convert* calls the progress function in a single sequence with p*rogress_percent* advancing from 0 to 100, *rscore_fconvert* does this for each file and then one additional time for the 2nd stage of processing. *progress_percent* takes the value zero exactly once for each file allowing the transition from one to the next to be detected. When every page has been processed *progress_percent* once again begins at zero, reaching a maximum close to 100 when the whole task is complete. If the callback should return false at any point the whole build is abandoned.

This arrangement allows the client application to implement a progress indicator for the whole process, incrementing only on the zero transitions. If desired a per-page progress indicator is also possible.

General information

Input images

ReadScoreLib accepts a wide range of input quality, resolution, exposure and geometric distortion. The image may be in colour, greyscale or monochrome. See below for more information on the preprocessing transformations applied

Image preprocessing

Before any music recognition takes place ReadScoreLib prepares each image as follows:

Preprocessing transformations	
Thresholding	Image is converted from colour, greyscale or monochrome to binary. The techniques used attempt to allow for variations in colour and shading
clipping	Material surrounding one or more areas of musical (for example in a device photograph) notation is stripped away
Linearisation	The image squared up: Distortions such as photographic distortions are reversed leaving staff lines straight and level
Segmented rotation	In some circumstances the image is broken into sections that are independently deskewed and dovetailed

The preprocessed image can be useful for some applications and is available using the *rscore_optLens* option.

When enabled the *rscore_optDeskewMonochrome* option causes monochrome pages to be subjected to segmented rotation instead of thresholding and linearisation

Image scale

For good accuracy ReadScoreLib requires an image where the vertical distance between staff lines is at least 9 pixels. The ideal range is 15 - 25

Operation abort

The calling process can cancel processing by returning false from the progress callback

Options

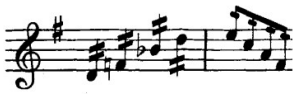
The options argument allows the caller to specify certain processing options as well as the application name and version as required for the XML identification element. The processing options are selected through bits set in the rscore_options::flags word (see below)

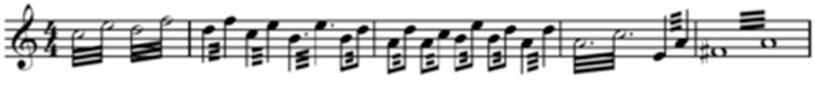
Option flags	
rscore_optExcludeUpperStaffs	Ignore all staffs above the lower two, taken system by system
rscore_optParseTremolo	recognise tremolo notations
rscore_optTremoloExpand	generate the equivalent non-tremolo notation for recognised tremolo NB leave flag unset to generate tremolo notation in output MusicXML
rscore_optXmlObjectBounds	Generate a text file listing the bounds of objects. The text file has the same name as the XML file but with the TXT extension
rscore_optXmlDefaultCoords	Generate MusicXML Default x, y attributes
rscore_optLens	write preprocessed image to the address supplied by the image_out parameter
*rscore_optText	Recognise text such as lyrics and markings

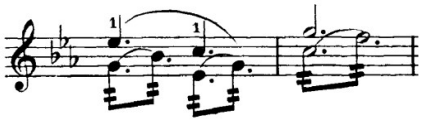
* Text OCR will be supported in the next release of ReadScoreLib

Tremolo

ReadScoreLib supports the following tremolo notations. Between one and four strokes are recognised.

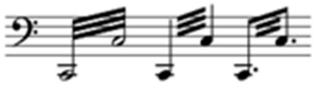
1) Stem crossing strokes 

2) two note tremolos 

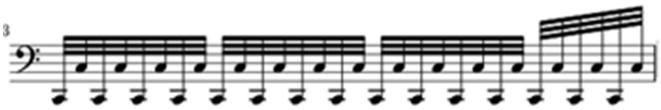
3) Alternative notations 

Expansion

Normally these notations generate the corresponding MusicXML object descriptions which will then render as the same tremolo notations when rendered. The *rscore_optTremoloExpand* flag can also be set causing ReadScoreLib to generate the same music but in the longhand equivalent. Thus



Would become



Establishing the location and bounds of musical objects

ReadScoreLib provides several features that allow an application to obtain the identity and location of recognised score objects. These include

1) Optional output text file containing the bounding boxes of all musical objects referenced against their MusicXML counterparts

2) Optional bounding box information written as comments inline with the MusicXML

3) The rscore_getbarinfo() API returning the bounding quadrilateral of barline bounded bars together with their beat count

Barline information

For applications that require only barline information (eg for score following) ReadScoreLib can be queried directly for the position and height of every barline in the score without recorse to the MusicXML or the text file. The *getbarcount* and *rscore_barinfo* APIs are used to obtain the positions and heights of barlines in the coordinates of the original page. If ReadScoreLib has transformed the geometry of the page (see *rscore_optLens*), then the transformations are reversed before being passed back.

int	rscore_getbarcount		Return the total number of bars in the input
	rscore*	rsc	handle to rscore instance

rscore_barinfo	rscore_getbarinfo(		
	rscore*	rsc	handle to rscore instance
	int	barindex	the index of the bar [0..rscore_getbarcount-1]

struct	rscore_barinfo		Barline information
	rscore_barline	opening_barline	position of opening barline
	rscore_barline	closing_barline	position of closing barline
	Int	startbeat	the starting beat number of this bar
	Int	beatcount	the number of beats in this bar
	Unsigned	flags	see rscore_barflags
	int	first_beat_offset	**percentage** of bar width to the left of the first note or rest

struct	rscore_barline		Barline dimensions
	rscore_point	base	the barline lower point
	int	height	The height in pixels

enum	rscore_barflags	Barline flags
	rscore_firstOfSystem	marks the leftmost bar in the system
	rscore_eoscore	marks the last bar in the score
	rscore_splitbarL	box encloses the part of the bar at the end of one system
	rscore_splitbarR	box encloses the part of the bar at the beginning of the next system
	rscore_firstOfSection	marks the first bar of a section, for example where one movement ends and another begins on the same page

Errors

Error status is normally reported through a passed-in *rscore_errorinfo* struct giving details of the error (see rscore.h for details).

Version information can be retrieved through the *rscore_version* struct.

Supported platforms

ReadScoreLib libraries are available for Windows, Mac, Android and iOS. A Linux version will be available shortly.