ReadScoreLib Optical Music Recognition Library

(documentation relates to RSL 4.3.X.X)

ReadScoreLib from Organum 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 full music rendering and MusicXML query capabilities.

ReadScoreLib is designed to cope with a wide range of engraved and printed notation from 19th Century engravings to those produced by modern score writing software. It supports published score images, scanned scores as well as photographically distorted images photographed with a device camera. ReadScoreLib does not support handwritten music or novelty fonts such as Real Books, or special notation dialects such as Shape notes and colour notes.

The performance of the library varies according to platform but on an average with an iPhone 11 a page of music takes 1-2 seconds to process.

ReadScoreLib applies musical rules and can compensate for missing elements such as time signatures, rests and tuplet marks, all of which are common in printed music. The library can also deal with variable size systems, varying staff gauges, partial staves and transposing instruments. The MIDI files generated by ReadScoreLib play naturally and continuously and without the jerks and jumps which is sometimes seen in OMR output. ReadScoreLib is also the only OMR system to support the varied tremolo notations often found in printed music.

The PlayScore app

The PlayScore app uses ReadScoreLib and can be used to evaluate the library.

PlayScore 2 is available on the Apple App Store and on the Android Play Store. PlayScore 2 is kept up to date with the latest library.

Platform APIs

This documentation describes the C language interface to ReadScoreLib. The ReadScoreLib SDKs for iOS/macOS and Android provide APIs for Swift, and Java respectively. These APIs are equivalent to the C API described here. Their use will be apparent from the documentation below.

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*	data	array of 32bit pixels in (32bpp RGBA) format with the alpha channel (bits 24 – 31) should be zero
	int	imageWidth	the pixel width of the image
	int	imageHeight	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 specifying 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 a sequence of memory images

rscore_bconvert set of functions allow multiple memory-resident page images to be built into single MIDI and MusicXML files subject to selected options

rscore*	**rscore_bconvert_begin(**		**read one or more page image files and output corresponding MIDI and MusicXML files**
	rscore_progress_callback	cb	user-supplied callback called periodically during the build process 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 specifying selected options (see rscore_optionsflags)
Return	rscore_errorinfo		handle to rscore instance

rscore*	**rscore_bconvert(**		**read one or more page image files and output corresponding MIDI and MusicXML files**
	rscore*	rsc	The rscore instance handle returned from rscore_bconvert_begin
	const unsigned*	data	array of 32bit pixels in (32bpp RGBA) format with the alpha channel (bits 24 – 31) should be zero
	int	imageWidth	the pixel width of the image
	int	imageHeight	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
Return	rscore_errorinfo		error information

rscore*	**rscore_bconvert_end(**		**read one or more page image files and output corresponding MIDI and MusicXML files**
	rscore*	rsc	The rscore instance handle returned from rscore_bconvert_begin
	const char*	boundsFilePath	If non-NULL the full path to which object bound information will be written (see rscore_optXmlObjectBounds)
	const char*	midiFilePath	if non-null the full path to which the MIDI file will be written
	const char*	xmlFilePath	if non-null the full path to which the MusicXML file will be written
Return	rscore_errorinfo		error information

The three rscore_bconvert functions allow multi-page music to be built into MIDI and MusicXML files. These functions can also be used to create a text file specifying the coordinates of objects within the images.

When a series of images is built using the rscore_bconvert functions they are built as continuous music rather than as a series of pages. As new files are submitted with rscore_bconvert a parse tree is built for each page. When rscore_bconvert_end is called the sequence is converted to a linear relational format. This representation is continuous over pages so that cross-page features such as part bars, slurs and ties can be translated faithfully. At this point also, statistical information is extracted and applied globally to improve general recognition accuracy. Finally MusxicXML is generated from the internal relational representation.

Since any sequence of pages submitted between rscore_bconvert_begin and rscore_bconvert_end will be built into a continuous MusicXML stream, it is best to break the music up into individual movements and to submit these separately. For each sequence, rscore_bconvert_begin is called first. This establishes any progress/cancellation callback and returns a handle to the ReadScoreLib for the present session.

The rscore handle is then used to submit pages of music in sequence via rscore_bconvert. This function takes the same image parameters as the single-page rscore_convert API described above. See description above for details. The rscore_error return may be checked for status information relating to the image.

When all pages have been submitted through rscore_bconvert, rscore_bconvert_end should be called to complete the build. This function takes the paths to desired MIDI, MusicXML and bounding-box-ID file.

Progress callback

ReadScoreLib processes music in two stages Rather than treat each page as a separate unit, *rscore_bconvert* creates a syntax tree for the whole input, page by page as described above.

Because of this organisation the progress callback functions for *rscore_convert* and *rscore_bconvert* work differently. Whereas *rscore_convert* calls the progress function in a single sequence with p*rogress_percent* advancing from 0 to 100, *rscore_bconvert* does this for each file and then one additional time when rscvore_bconvert_end is called for the final 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, or perhaps a few times during each rscore_convert call. Since the number of pages will be known the progress indicaror can be arranged to asvance smoothluy from 0 to 100. 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. For good results the scale of the submitted bitmaps should be such that the distance in pixels between individual staff lines is at least 10. For best results this distance should be between 15 and 24. Larger scales are possible but performance deteriorates as the square of linear distance. If the image is clear a metric of 16 will often give good results at two or three seconds per page on an iPhone 6.

The table below gives suitable ranges for image capture from music using different capture methods. For best results for a particular type of music, experiment with any parameters under your control such as light and scale.

Device	Scale	Comments
Scanner	Scan at 280 – 310DPI	
PDF to JPG converters	Set converter to 300 – 500DPI	It is worth trying a wide range of conversion DPI, especially where the image quality appears poor
Photography	Aim for a staff metric between 14 and 18	Take image in good light and aim for a minimum of spherical and other photographic distortion.
From a book		Ensure that the page lies as flat as possible. The left side especially, where the system line and the signatures are should be clear
PDF and other images created by score writers such as Sibelius, Finale, Dorico etc	Convert from PDF at 150 – 300DPI	Ensure that the image is dark enough to show features such as staff lines and stems clearly. These images can sometimes be bright and have thin lines. This can cause poor results

The image may be in colour, greyscale or monochrome. Greyscale is usually better as ReadScoreLib can choose its own threshold at different areas of the image. 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. Music parsing and MusicXML generation options are selected through bits set in the rscore_options::flags word (see below)

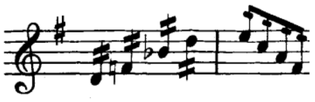
Certain flags such as `rscore_optParseTremolo` can be used to reduce the risk of recognition false positives by suppressing parsing for particular constructs. Normally these flags should be set unless the music is known to be free of a particular construct.

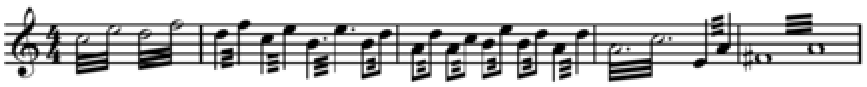
Option flags	
rscore_optParseTremolo	recognise tremolo notations (flag now has no effect as tremolo support is permanently enabled)
rscore_optXmlDefaultCoords	Generate MusicXML Default x, y attributes
rscore_optXmlObjectBounds	Generate a text file listing the bounds of objects. The text file has the same name as the XML file but with the TXT extension
rscore_optTransposingInstruments	Guess transposing instrument transpositions where possible from key signature
rscore_optFrenchTimeSignatures	Recognise French time signatures (those engraved in parentheses above the barline)
rscore_optLens	write preprocessed image to the address supplied by

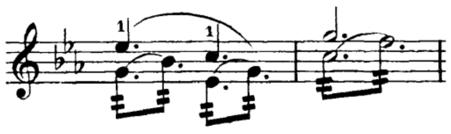
	the image_out parameter
rscore_optLayout	include MusicXML print elements to reflect the system and page boundaries that were found in the score. This will preserve the score's page and system layout exactly.
rscore_optDark	Suppress RSL's normal adaptive thresholding and instead perform a simple 50% threshold. This can sometimes be useful for dark images. To suppress all thresholding provide pre-thresholded images.
rscore_optSuppressOptimisation	Disable the false relation feature
rscore_optMIDISwing	MIDI only: Apply the wing music convention to the MIDI output
rscore_optMIDISplitStaff	MIDI only: direct the MIDI output for each staff into two channels rather than one. The channels correspond to a stem-down and a stem-up part

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 

Transposing instruments

Normally ReadScoreLib assumes all instruments untransposing and makes use of all key signature information in establishing the key. If the `rscore_optTransposingInstruments` flag is set this assumption is no longer made and transposing instruments are where possible identified from their key signatures. In these cases the appropriate transpositions are written into the MusicXML file and the correct key signature specified. However note that not all transposing instruments can be recognised in this way. These are of two types:

1) Octave transpositions such as the piccolo and the double bass.
2) Instruments such as horn and trumpet. These are normally written without key signature and cannot be deduced.

In addition to the `rscore_optTransposingInstruments` flag, `rscore_options::isosig` (lowest setting 1) can be used to define the number of staves, counting from the bottom that will be presumed to have the same key signature. The larger isosig the greater the amount of redundancy RSL can take advantage of, and therefore the more accurate the recognition of scores where the image quality is poor.

ReadScoreLib will shortly be providing support for OCR text (see below). This will include instrument names, thereby allowing more comprehensive support for transposing instruments.

Split staff

The MIDI generated by RSL is divided into channels with one channel per staff. When `rscore_optMIDISplitStaff` is set two channels are allocated to every channel. With this setting the music in each staff is conceived as being in two voices which are directed to the two channels, the lower voice having the lower channel. This affects only MIDI output. The division into channels in no way affects the voice assignment in the MusicXML output.

Swing

Swing is a performance convention in which the length of certain notes and rests is altered to create the swing effect. Swing does not affect how the music looks, just how it sounds and applies in RSL only to MIDI output. Swing MIDI is generated when the `rscore_optMIDISwing` flag is set.

Optimisation

ReadScoreLib uses a number of mechanisms for reconstructing missing or indistinct musical constructs arising from poor score quality or photography. Several of these attempt to correct missing, occluded or otherwise doubtful accidentals. One of these methods although generally helpful can create unwanted effects in some music. This optimisation looks for cases where notes on the same beat in different voices, or within a chord on a single voice lie on the same degree of scale but have different accidental modification. This usually happens because an accidental is missing in the score, or one is recognised incorrectly for some reason, and generally improves accuracy. However in some music *false relations* as they are known are intentional. They are not uncommon in Mozart for example. For this reason the feature can be suppressed with the `rscore_optSuppressOptimisation.`

It may prove best to suppress the feature for good quality scores of classical music. For most other kinds of music the incidence of missing or misrecognised accidentals is generally greater than that of intentional false relations and accuracy will be better with the feature left on.

Establishing the location of objects

ReadScoreLib provides several features that allow an application to obtain the identity and location of recognised score objects. The requirements of such a system go beyond the scope of MusicXML. MusicXML does support a *Default x/y* attribute type but these are unsuitable because they apply to some objects only and supply only a single pair of coordinates, rather than full bounding box information. Moreover *Default x/y* is a hint facility and if used rigidly to specify the coordinates of objects on the original page unpredictable behaviour is likely when the music is rendered. ReadScoreLib can generate *Default x/y* but the XML produced is intended for use as a key into the text file and not as an aid to rendering.

As MusicXML cannot itself contain the information required to locate and bound objects in the original image, ReadScoreLib writes this information to a separate file. The pathname to the file is specified in the rscore_bconvert_end API. In addition, the same information is added as comments to the MusicXML file.

The present version (3.1) of MusicXML lacks a comprehensive system of unique identifiers for objects. ReadScoreLib therefore provides an alternative mechanism based on the *Default x/y* attribute, used as a key into the text file. Alternatively if XML comments can be read by the XML client, bounding box information can be accessed directly.

When *rscore_optXmlObjectBounds* is set ReadScoreLib generates bounding box information in both forms, as comments in the XML file and as a separate text file. If the XML client cannot read these comments in the course of reading the object itself, the *rscore_optXmlDefaultCoord* flag can be used to select *Default x/y* attribute generation. The application software can then use this as a key into the corresponding bounding box information in the text file. As it reads an object (eg a rest), the MusicXML client (the XML parsing software) will read the *Default x/y* attribute. The coordinate information can then be used to look up the bounding box information in the text file. To make key duplication unlikely *Default x/y* attribute is given a five digit floating point precision. The two digits after the decimal point are randomised slightly to provide a unique key without affecting the positioning of objects when rendered.

NB Default x/y attributes sometimes give rise to unexpected results in rendering software as they reduce the client's ability to format a score freely. It is generally better to run RSL separately with rscore_optXmlDefaultCoord set specifically for the purpose of generating coordinates, and otherwise to leave it unset.

The bounding-box-ID text file has a simple format consisting of one line of text for each object.

<barnumber>_<ref-x>_<ref-y>: <object-name> (<pagenumber>, <bounds>)

Where <ref-x>_<ref-y> is the *Default x/y* attribute referencing the MusicXML file. Where an XML construct does not carry *Default x/y* (-1, -1) is substituted.

For example the following represents a clef found on page 5 (numbered from 1) enclosed within a box whose bottom-left and top-right corners lie at (584,2735) and (612,2797)

```
2_51.001_406:clef (5,584,2735,612,2797)
```

Barlines, rests, accidentals and text (dynamics), ties and slurs follow the same format.

Some objects specify coordinates adapted to their positioning as musical objects. A note head for example specifies its centre only, as other dimensions can be deduced from the staff gauge.

```
<barnumber>_<ref-x>_<ref-y>:head (<pagenumber>, <point-xy>)
```

The sequence of entries in the text file reflects music syntax. A stem on its own is represented by a “stem” entry followed by one or more head entries. Where stems are beamed a “group” entry appears first. The following sequence is the first “group” sequence from plolnaise.txt as derived from the polonaise.jpg example.

```
1_-1_-1:group (1,525,2314,750,2516)
1_-1_-1:stem (1,556,2473,556,2318)
1_254.01_-154.99:head (1,545,2304)
1_254.01_-86.986:head (1,545,2372)
1_-1_-1:group (1,647,2384,671,2436)
1_368.01_-48.984:rest (1,647,2384,671,2436)
1_-1_-1:stem (1,747,2516,747,2386)
1_445.02_-85.982:head (1,736,2373)
1_445.02_-47.98:head (1,736,2411)
1_445.02_-18.978:head (1,736,2440)
```

The rest in this example is the rest between the beamed stems.

In scores where systems are not all the same size, a special entry makes it possible to determine the physical staff for a MusicXML part

```
<barnumber>_<ref-x>_<ref-y>: staffspace(<pagenumber>, <staff-gauge>, <staff-count>,
<staff-index>)
```

As already explained, <ref-x> and <ref-y) are not intended to specify a location in space, but to provide a unique link between MusicXCML and The bounding-box-ID text file as described above.

Barline (measure) 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 recourse 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.

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.

Legal information

Music copyright

Much music is subject to copyright. Where copyright applies it is the licensee's responsibility to establish whether a particular use of ReadScoreLib is legal.

Liability

ReadScoreLib is supplied *as is*. Organum Limited and Dolphin Computing Limited make no warranty as to the conformance of ReadScoreLib to any particular specification or the suitability of ReadScoreLib for any particular purpose.

Organum Limited and Dolphin Computing Limited shall not be liable in any way for loss, cost, injury or harm that may result from use of ReadScoreLib

Licensed features

This documentation describes all ReadScoreLib features. Licensees should consult their particular license agreement for details of which features are covered.

Appendix 1 – Deprecated API

Recognition from a sequence of memory images

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.

Appendix 2 – Supported musical symbols

RSL supports

- Bars, notes, rests, accidentals including double accidentals, cancelling accidentals and cautionary accidentals
- Braces groups of staves representing a single instrument (eg piano) with multi-staff stems, cross-staff beaming and over-barline beaming
- Tuplets: triplets, duplets, quintuplets, septuplets etc (both marked and implied)
- Staff bracketing: grand staff braces, grouped staff brackets etc
- Measures: bar lines, double bar lines, repeats, 1st and 2nd ending, bars spanning systems and pages

[1] JPG only

- Anacruses, compliment anacruses
- Dynamics: f, ff, fff, fz, fp, mf, p, pp etc
- Hairpins (crescendos and diminuendos)
- Articulation (>, ^, . –, portato etc)
- Ornaments trills, turns, mordents, shakes, spread chords, + etc
- Tremolo: note strikethrough, alternating, beamed alternating white notes etc
- Special symbols: fermata, repeat-bar, Ottava 8ve etc
- Fingering for piano, violin etc
- Slurs and ties
- Clefs (system and inline): neutral[2], treble, bass, tenor, alto, soprano etc including octave shift variants
- Key changes: system, inline, cautionary
- Time signatures: system, inline, cautionary and implied

[2] The percussion clef is supported together with crossheads etc. This is included in the MusicXML output but not MIDI.