REX SDK

DOCUMENTATION

VERSION 1.9

CONTENTS

1. INTRODUCTION 4

1.1 WHO ARE YOU? 4
1.2 RECYCLE FILE FORMATS 4
1.3 THE REX API 4
1.4 REX COMPLIANCE TEST 4

2. CONCEPTS 6

2.1 SLICES 6
2.2 API SUBSETS 6
2.3 REX OBJECTS 6
2.4 EXAMPLE CODE 6
2.5 DLL FOR TESTING 6

3. HOW TO:S 7

3.1 TYPE DEFINITIONS 7
3.2 COMPILING REX.H 7

4. API DETAILS 8

4.1 OVERVIEW 8
4.1.1 Functions 8
4.1.2 Data Types 10
4.1.3 Error Codes 10
4.1.4 Multi Threading 10
4.2 BASIC FUNCTIONS 11
4.2.1 REXInitializeDLL 11
4.2.2 REXUninitializeDLL 11
4.3 SLICE RENDERING 12
4.3.1 REXCreate 12
4.3.2 REXDelete 13
4.3.3 REXGetInfo 13
4.3.4 REXGetInfoFromBuffer 14
4.3.5 REXGetCreatorInfo 15
4.3.6 REXGetSliceInfo 16
4.3.7 REXSetOutputSampleRate 17
4.3.8 REXRenderSlice 18
4.3.9 REXCreateCallback 19
4.4 PREVIEW PLAYING 20
4.4.1 REXStartPreview 20
4.4.2 REXStopPreview 21
4.4.3 REXSetPreviewTempo 21
4.4.4 REXRenderPreviewBatch 22
4.5 DATA STRUCTURES 23
4.5.1 REXInfo 23
4.5.2 REXCreatorInfo 24
4.5.3 REXSliceInfo 24

5. APPENDIX A – FILE FORMAT ATTRIBUTES 25

6. APPENDIX B – SYSTEM REQUIREMENTS 26

6.1 WINDOWS 26
6.2 MAC OS X 26

7. APPENDIX C – RECYCLE'S CLIPBOARD SUPPORT 27

7.1 COPY LOOP 27
7.2 PASTE AS NEW DOCUMENT 27
7.3 CLIPBOARD FORMATS 27

8. APPENDIX D – REX COMPLIANCE TEST PROTOCOL 28

8.1 LOADING FILES 28
8.2 IMPORT AND PLAYBACK 28
8.3 PREVIEW PLAY 29

1. INTRODUCTION

1.1 Who Are You?

There are two intended audiences for this text: those who wish to know specifically what REX2 files and the REX API are and those who have to know the exact and gory details of including REX support in their software products, namely software developers. If you consider yourself to belong to the first category, read chapter 1 - 3 and skip the rest. If you're a developer, I'm afraid that you'll have to read the whole lot, and carefully at that.

All people reading this document are assumed to be familiar with the Propellerhead ReCycle software, the concepts used in that application, as well as what you can do with it. If you are not familiar with it, you had better download the demo from our web site (http://www.propellerheads.se/downloads/) and trying that out before you read any further.

Developers are assumed to be confident using the C language.

1.2 ReCycle File Formats

Before ReCycle 2.0, there were two file formats associated with ReCycle: ReCycle Document files (with the extension .RCY on Windows systems) and ReCycle Export files, popularly called "REX"-files, because of its ".REX"-extension on Windows systems. The ReCycle Document Files could be saved and then re-edited in ReCycle, whereas the REX files could only be exported, once and for all, and then imported into Cubase VST.

With the release of ReCycle 2.0, both of these file formats became obsolete; they left the stage to make way for a new format, *REX2*, which has the properties of both of its predecessors. REX2 files are the new native ReCycle Document files, and can as such be loaded, saved and reloaded into ReCycle for continued editing and fine-tuning. REX2 files can be imported for use in other applications by means of the *REX API*.

Besides containing all the new information that ReCycle 2.0 needs, REX2 files also have the property of being compressed (in the "zip" kind-of way, not the musical one). This is accomplished using a proprietary *non-lossy* algorithm which manages to reduce the size of the files by around 40-60% in most cases, without compromising sound quality in any way. Unpacking a REX2 file produces sound data that is bit-for-bit identical to that contained in the original file.

1.3 The REX API

The REX API provides the necessary functions for using REX2 files in your application. It is distributed as a *Dynamic Link Library* (DLL on Windows, framework on Mac), which must be installed with your application. DLLs are a technique that is highly platform-dependent, which means that there is a different DLL file for each supported platform. We currently support Windows and Mac OS X platforms. By licensing the REX SDK, you have been granted the right to distribute and install the DLL files together with your application.

The REX DLL/framework should not be installed in system folders. On Windows, install it in your applications folder, on Mac, put it in the Frameworks folder of your app bundle.

Through the API your application is enabled to make use of the information in REX2, as well as old-style REX- and ReCycle Document (".RCY") files. This is the reason the API is called the "REX API" and not the "REX2 API". Everything you can do with a REX2 file by means of the API, *you can also do with a REX or RCY file*. In the rest of this document we will refer to to all of these files as "REX2", just for the sake of simplicity.

1.4 REX Compliance Test

To obtain permission to include references to Propellerhead and REX or to use the REX logotype in its merchandise and marketing material, a REX licensee has to run the application, before its release, through a series of tests in order to determine whether the application complies with the REX SDK and what is expected of

a REX capable application. In addition to ensuring a consistently pleasant experience for users of REX files, this test provides you as a developer with a way to exercise different parts of your application and to reveal potential problems. Though the REX API may be small and easy to use, the technology is actually quite deep.

The *REX Compliance Test Protocol*, available in Appendix D of this document, describes the test procedure, which involves a set of files that you must try to use with your application. The necessary files are provided with this SDK and have been picked to test different aspects of the REX API. If your application can load all of these files without problems, you can be certain that your implementation works reliably. Some of the test files are even meant to provoke errors, so that you can verify that your application handles these circumstances correctly.

To be considered REX compliant, your program must be able to handle all of these files as expected, according to the test protocol. An application cannot claim to support the REX file format or to be REX compliant unless it has undergone this test for each major release of the software, i.e. it is required for 1.0 and 2.0 but not for 1.0.1 or 1.5.

2. CONCEPTS

2.1 Slices

ReCycle divides a chunk of sampled sound into one or more *slices*. These can be manipulated in the program, and then exported to one of the samplers or saved in one of the file formats that ReCycle supports. When exporting only the slices which are *in between the loop locators* are exported. Those are the slices that are accessible through the REX API.

2.2 API Subsets

The REX API provides two disparate areas of functionality: *Slice Importing* and *File Previewing*. To make use of a REX2 file you have to *import* each and every one of its slices into your application. However, before the slices are imported, you may wish to let the user choose among a bunch of REX2 files (for example on a sample CD) via an ”Open File” dialog or similar. In that case, the REX API also provides a set of functions to let you play back a *preview* of the file.

2.3 REX Objects

To do anything useful with a REX2 file, you need to let the API transform it into a *REX object*. All actions are then carried out on the REX object. You could easily create a wrapper class for REX objects in C++, if you wish.

Creating a REX object is done by first loading the REX2 file into a memory buffer and then calling a function in the REX API on that buffer. Out comes a handle to a REX object that you need to keep until you’ve got all slices you need or are finished previewing the file.

2.4 Example Code

A test program is included in the SDK. It is intended that you steal as much as you can from this example, so it is very simple, yet accurate. All it does is extract the slices in a REX2 file and save each one as a separate ”.WAV”-file.

The example is provided in ANSI C compliant source code. There are two functions in the example that you should take a closer look at: `ExtractAllSlices()`, which does the actual work, and `GetErrorText()`, which trans-forms a REX API error code into an understandable line of text that you could present to the user if anything goes wrong.

There are a couple of things that the source code does not demonstrate: how to preview a file, how to let the user abort `REXCreate()` (although that is shown pretty clearly in the documentation on that function) and how to use the PPQ information that can be extracted for each slice. These things are supposed to be simple enough for you to figure them out for yourself. If not, ask us!

2.5 DLL for Testing

The SDK contains two versions of the REX Shared Library for each platform: the *Deployment* version and the *Testing* version.

While developing your REX-capable application, you should use the *Testing* flavoured DLL, which is littered with bug-checks to make sure that nothing fishy slips through into the shipping code. Depending on how you work, you could choose to ship the *Testing* version of the DLL to your beta testers, or use the *Deployment* version. In any case, make sure that you don’t ship the *Testing* version with your final product, since it is quite a bit slower and larger than the optimized *Deployment* version. The functions in the REX API which do not return error codes will produce TRACE outputs in case something is wrong, so keep an eye on the debug output window in your IDE!

3. HOW TO:S

3.1 Type Definitions

The REX API makes use of the type REX_int32_t which needs to be a 32-bit signed integer type. The REX.h header file will try to determine a suitable type for compilers that it knows about but if you are using another compiler you will have to define this type yourself as below before including REX.h.

```
#define REX_TYPES_DEFINED 1          // prevent REX.h from trying to figure out types automatically
typedef int REX_int32_t;             // On this platform int is a signed 32-bit integer.
#include “REX.h”
```

3.2 Compiling REX.H

”REX.H” needs to know what platform for are compiling for and requires REX_MAC or REX_WINDOWS preprocessor macros set.

```
// macOS
#define REX_MAC 1
#define REX_WINDOWS 0

// Windows
#define REX_MAC 0
#define REX_WINDOWS 1
```

API DETAILS

This chapter contains the details on the REX API.

3.3 Overview

The API is really very simple. Before reading this, take a look at REX.h, which is probably enough information to start using the API. However, before you start assuming anything, it is wise to read through the section on each function that you intend to use.

3.3.1 Functions

The API functionality is divided into three distinct parts:

1. The *Basic* functions initialize the DLL:

Function	Short Description
`REXError REXInitializeDLL()`	Initialize the REX DLL. Direct call to DLL/dylib when using the .framework bundle on Mac (the default and recommended usage on Mac).
`REXError REXInitializeDLL_DirPath()`	Load and initialize the REX DLL found in directory iDirPath. Call into dynamic loader REX.c (the default and recommended usage on Windows).
`void REXUninitializeDLL()`	Uninitialize the REX DLL.

2. The *Slice Rendering* functions are used to extract individual slices from a file that resides in memory:

Function	Short Description
`REXError REXCreate(REXHandle*, …)`	Create a REX object from a file in RAM.
`void REXDelete(REXHandle*)`	Destroy a REX object.
`REXError REXGetInfo(REXHandle, …)`	Get information about a REX object.
`REXError REXGetInfoFromBuffer(…)`	Get information from a file in RAM.
`REXError REXGetCreatorInfo(REXHandle, …)`	Get optional information about a REX object.
`REXError REXGetSliceInfo(REXHandle, …)`	Get information about a specific slice.
`REXError REXSetOutputSampleRate(REXHandle, …)`	Set the desired sample rate for the sample data ouput by `REXRenderSlice()` and `REXRenderPreviewBatch()`.
`REXError REXRenderSlice(REXHandle, …)`	Render a slice in the given buffer.
`REXCallbackResult REXCreateCallback(REX_int32_t percentFinished)`	User-defined callback function.

3. The *Preview* functions are used to preview-play a REX object:

Function	Short Description
`REXError REXStartPreview(REXHandle, …)`	Starts preview of a REX object.
`REXError REXStopPreview(REXHandle, …)`	Stops preview.
`REXError REXRenderPreviewBatch(REXHandle, …)`	Render a number of samples in a buffer.
`REXError REXSetPreviewTempo(REXHandle, …)`	Change tempo during preview.

3.3.2 Data Types

Three data structures and three simple types belong to the API:

Type	Short Description
`REXInfo`	Data structure containing basic information about a REX2 file.
`REXCreatorInfo`	Data structure containing optional information about a REX2 file.
`REXSliceInfo`	Data structure with information about one particular slice.
`REXError`	Error code. Always compare an error with a specific constant (see below).
`REXHandle`	Handle of a REX object, created from a file in memory. This is a pointer-sized integer value the API uses to identify the object. A zero handle means that it is invalid.
`REXCallbackResult`	Result code from the `REXCreateCallback()` user callback.

3.3.3 Error Codes

Almost all functions return an error code that indicates whether the operation was successful or not. In case of success, `kREXError_NoError` is returned. This is regardless of what function was called. If something has gone wrong, the error code will tell you what the problem is (check the documentation for each function to see which error messages it may return), and you should then do as you normally do when using an API, clean up and recover as gracefully as possible.

It is important that you always test explicitly against `kREXError_NoError` after each function call, and *not* against zero or something semi-arbitrary like that.

In the example code supplied with the API there is a function `GetErrorText()`, which takes an error code and outputs a string that could be presented to the user in a dialog box etc. We encourage you to grab this function and mutilate it in whatever way you find suitable for your application (for example, you could make it load the texts from a set of resource strings if your application supports multiple languages).

The error codes are divided in two major categories: *run-time errors* and *implementation errors*. The implementation errors will not occur if your program conforms exactly to this SDK documentation and are distinguished through their names so that you can determine what problems could be easily fixed because there is something in the API that has been misinterpreted, and what errors are to be expected in a running application.

Please see the file REX.h for a complete list of error codes.

3.3.4 Multi Threading

The REX API is *not* thread-safe. Only call the functions of the DLL from one thread at a time, or you will be in serious trouble. An exception from this rule is the preview-API function `RenderBatch()`, which is guaranteed to work when called from a different thread than the rest of the functions.

3.4 Basic Functions

3.4.1 REXInitializeDLL

Initialize the REX DLL. Must be called before any other function is called.

```
REXError  REXInitializeDLL()
```

Return Value	Description
`kREXError_NoError`	Success!
`kREXImplError_DLLAlreadyInitialized`	`REXInitializeDLL()` has already been called successfully.

Once `REXInitializeDLL()` has been called, you should call `REXUninitializeDLL()` when exiting your application.

3.4.2 REXUninitializeDLL

Unloads the previously loaded REX DLL.

```
void      REXUninitializeDLL()
```

Only call `REXUninitializeDLL()` once.

3.5 Slice Rendering

The slice rendering API is the heart of the REX API. The DLL must have been loaded, using `REXInitializeDLL()`, for these functions to be available.

3.5.1 REXCreate

Creates a REX object from a file buffered in memory.

```
REXError  REXCreate(REXHandle* handle,
                    const char buffer[],
                    REX_int32_t size,
                    REXCreateCallback callbackFunc,
                    void* userData);
```

Argument	Description
`handle`	Address of a handle to a REX object. This will be filled with the handle of the new REX object, if it was successfully created, or zero otherwise.
`buffer`	Address of the memory-buffered REX, REX2 or RCY file. The buffer can be deallocated as soon as you have a valid handle to a REX object.
`size`	Size of the buffered file.
`callbackFunc`	Function which will be called regularly while the REX object is being created. The parameter may be NULL if no callback is needed.
`userData`	Optional user-defined data that will be supplied to the callback function, if that is used.

Return Value	Description
`kREXError_NoError`	Success!
`kREXError_OutOfMemory`	The REX object cannot be created because of low memory.
`kREXError_FileCorrupt`	The file in "`buffer`" with the size "`size`" is not a valid REX, REX2 or RCY file.
`kREXError_REX2FileTooNew`	This REX2 file cannot be loaded because it is too new to be compatible with this version of the API.
`kREXError_OperationAbortedByUser`	The operation was aborted by the user callback.
`kREXError_FileHasZeroLoopLength`	This *RCY* file cannot be loaded, because the "Bars" and "Beats" settings have not been set up. The PPQ length of the loop cannot be determined.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidArgument`	Either the "`handle`" or "`buffer`" pointer is NULL or "`size`" is invalid.

This function unpacks the whole file into an internal a memory buffer, so it may use a significant amount of memory and take some time. If the function returned `kREXError_NoError`, then the object "`handle`" refers to must be deleted later on, using `REXDelete()`.

3.5.2 REXDelete

Destroys a REX object.

```
void      REXDelete(REXHandle* handle);
```

Argument	Description
`handle`	*Pointer* to the handle of the REX object to destroy. Is zeroed by the function.

The REX object is destroyed and all its resources are deallocated. Make sure that you are not trying to preview or do other things with the object at the time this function is called, or afterwards. You should not try to delete a REX object more than once.

3.5.3 REXGetInfo

Get information about a REX object.

```
REXError  REXGetInfo(REXHandle handle,
                     REX_int32_t infoSize,
                     REXInfo* info);
```

Argument	Description
`handle`	Handle of the REX object to get information about.
`infoSize`	The expected size of the `REXInfo` structured which will be filled in by the function. Always set this parameter to `sizeof(REXInfo)`.
`info`	Pointer to a `REXInfo` structure which will be filled with information after successful completion of the function.

Return value	Description
`kREXError_NoError`	Success!
`kREXError_OutOfMemory`	Unable to get information because of low memory.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidHandle`	The handle does not refer to a REX object currently in memory.
`kREXImplError_InvalidSize`	The `infoSize` parameter has not been set up correctly. Always set this parameter to `sizeof(REXInfo)`.
`kREXImplError_InvalidArgument`	The "`info`" pointer is NULL.

If you use this function correctly, it may only fail because of low memory.

3.5.4 REXGetInfoFromBuffer

Get information about a file buffer in memory. Using this function you don't need to create a REX object before you can find out the tempo of a file etc. This could be useful in file-browsing situations, where fast GUI response is preferred and calling `REXCreate()` takes too long.

```
REXError  REXGetInfoFromBuffer(REX_int32_t bufferSize,
                               const char buffer[],
                               long infoSize,
                               REXInfo* info);
```

Argument	Description
`bufferSize`	Size of the buffered file.
`buffer`	Address of the memory-buffered REX, REX2 or RCY file.
`infoSize`	The expected size of the `REXInfo` structured which will be filled in by the function. Always set this parameter to `sizeof(REXInfo)`.
`info`	Pointer to a `REXInfo` structure which will be filled with information after successful completion of the function.

Return value	Description
`kREXError_NoError`	Success!
`kREXError_OutOfMemory`	Unable to get information because of low memory.
`kREXError_FileCorrupt`	The file in "`buffer`" with the size "`bufferSize`" is not a valid REX, REX2 or RCY file.
`kREXError_REX2FileTooNew`	This REX2 file is too new to be compatible with this version of the API.
`kREXError_FileHasZeroLoopLength`	No information can be retrieved about this *RCY* file, because the "Bars" and "Beats" settings have not been set up. The PPQ length of the loop cannot be determined.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidSize`	The `infoSize` parameter has not been set up correctly. Always set this parameter to `sizeof(REXInfo)`.
`kREXImplError_InvalidArgument`	Either the "`buffer`" or "`info`" pointer is NULL or "`bufferSize`" is invalid.

3.5.5 REXGetCreatorInfo

Get optional information about a REX object. The creator of a REX2 file *may* supply additional information (his/her name, URL, E-mail address etc), which can be extracted using this function. Please note that this information is not required to be present in a REX2 file, so be prepared to handle that condition. In particular, old-style REX and RCY files *never* contain optional information.

```
REXError  REXGetCreatorInfo(REXHandle handle,
                            REX_int32_t creatorInfoSize,
                            REXCreatorInfo* creatorInfo);
```

Argument	Description
`handle`	Handle of the REX object to get information about.
`creatorInfoSize`	The expected size of the `REXCreatorInfo` structured which will be filled in by the function. Always set this parameter to `sizeof(REXCreatorInfo)`.
`creatorInfo`	Pointer to a `REXCreatorInfo` structure which will be filled with information after successful completion of the function.

Return value	Description
`kREXError_NoError`	Success!
`kREXError_NoCreatorInfoAvailable`	There is no optional information available for this REX2 file. This is always the case for old REX and RCY files.
`kREXError_OutOfMemory`	Unable to get information because of low memory.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidHandle`	The handle does not refer to a REX object currently in memory.
`kREXImplError_InvalidSize`	The `creatorInfoSize` parameter has not been set up correctly. Always set this parameter to `sizeof(REXCreatorInfo)`.
`kREXImplError_InvalidArgument`	The "`creatorInfo`" pointer is NULL.

If used correctly, this function may fail only because the file does not contain any optional information, or because of low memory.

3.5.6 REXGetSliceInfo

Get information about a specific slice in a REX object.

```
REXError  REXGetSliceInfo(REXHandle handle,
                          REX_int32_t sliceIndex,
                          REX_int32_t sliceInfoSize,
                          REXSliceInfo* sliceInfo);
```

Argument	Description
handle	Handle of the REX object to get information about.
sliceIndex	Zero-based index of slice to get information about.
sliceInfoSize	The expected size of the REXSliceInfo structured which will be filled in by the function. Always set this parameter to sizeof(REXSliceInfo).
sliceInfo	Pointer to REXSliceInfo structure, which will be filled in with slice information after successful completion of the function.

Return value	Description
kREXError_NoError	Success!
kREXError_OutOfMemory	Unable to get information because of low memory.
kREXImplError_DLLNotInitialized	The REX DLL has not yet been initialized.
kREXImplError_InvalidHandle	The handle does not refer to a REX object currently in memory
kREXImplError_InvalidSize	The sliceInfoSize parameter has not been set up correctly. Always set this parameter to sizeof(REXSliceInfo).
kREXImplError_InvalidSlice	"sliceIndex" is out of range. Use fSliceCount in the REXInfo struct to find out how many slices there are.
kREXImplError_InvalidArgument	The "sliceInfo" pointer is NULL.

If you use this function correctly, it may only fail because of low memory.

3.5.7 REXSetOutputSampleRate

Sets the sample rate of the data that will be output by the two render functions: `REXRenderSlice()` and `REXRenderPreviewBatch()`.

```
REXError  REXSetOutputSampleRate(REXHandle handle,
                                 REX_int32_t outputSampleRate);
```

Argument	Description
`handle`	Handle of REX object to render a slice from.
`outputSampleRate`	Sample rate to render at. Range is 11.025 kHz – 1.0 MHz, inclusive.

Return value	Description
`kREXError_NoError`	Success!
`kREXError_OutOfMemory`	The output sample rate cannot be changed because of low memory. Strange as this may seem, it can happen because of internal data structures being rebuilt.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidHandle`	The handle does not refer to a REX object currently in memory.
`kREXImplError_InvalidSampleRate`	"`outputSampleRate`" is out of range.
`kREXImplError_IsBeingPreviewed`	The REX object is currently being previewed. You must stop previewing it by calling `REXStopPreview()` before changing the output sample rate.

The REX object's output sample rate is originally set to the loaded file's sample rate. This means that if you want the slices or previews of multiple files (with perhaps different orignal sample rates) to be rendered at the same rate, you have to call `REXOutputSampleRate()` with the desired output rate, for each one of the loaded files.

A file's original sample rate can be determined by looking at the `fSampleRate` field of the `REXInfo` structure, following a call to `REXGetInfo()`.

Changing the output sample rate will change the length of the rendered slices returned in the `REXSliceInfo` structure after a call to `REXGetSliceInfo()`. Thus, after changing the output sample rate you must no longer use the sample lengths returned by previous calls to `REXGetSliceInfo()` to determine the needed buffer sizes for each one of the rendered slices.

`REXSetOutputSampleRate()` also affects the data output by `REXRenderPreviewBatch()`.

3.5.8 REXRenderSlice

Render a specific slice from a REX object into a buffer.

```
REXError  REXRenderSlice(REXHandle handle,
                         REX_int32_t sliceIndex,
                         REX_int32_t bufferFrameLength,
                         float* outputBuffers[2]);
```

Argument	Description
handle	Handle of REX object to render a slice from.
sliceIndex	Zero-based index of slice to render.
bufferFrameLength	Length in sample frames of the buffers to render into. If the buffer is bigger than what is required to render the entire slice, the rest of it will be zeroed (0.0f).
outputBuffers	Array of two float pointers to buffers that will receive the rendered data for the left (ouputBuffers[0]) and right (ouputBuffers[1]) channels. If the REX object is a mono file, ouputBuffers[1] *must* be NULL. The pointers may *not* point at the same buffer and, when rendering a stereo file, two pointers *must* be supplied. The output data ranges from –1.0 to 1.0, inclusive.

Return value	Description
kREXError_NoError	Success!
kREXError_OutOfMemory	The slice cannot be rendered because of low memory.
kREXImplError_DLLNotInitialized	The REX DLL has not yet been initialized.
kREXImplError_InvalidHandle	The handle does not refer to a REX object currently in memory.
kREXImplError_InvalidSlice	"sliceIndex" is out of range. Use fSliceCount in the REXInfo struct to find out how many slices there are.
kREXImplError_InvalidArgument	Something is wrong with one of the parameters. With the *Testing* version of the DLL, a debug string has been output to help you detect what the problem is.
kREXImplError_BufferTooSmall	The buffer size (indicated by "bufferFrameLength") is not big enough to render the entire slice. The slice has not been rendered. The supplied buffer has not been touched.

The supplied buffer must be large enough for the entire slice. The required size of the buffer should be determined by checking the fSampleLength field in the REXSliceInfo for the slice. Please note that the sample length is dependent on the output sample rate that has been set for the REX object. The output sample rate originally is that of the file being loaded but can be changed by a call to REXSetOutputSampleRate().

3.5.9 REXCreateCallback

User-defined callback function, which is called regularly during creation of a REX object.

```
REXCallbackResult REXCreateCallback(REX_int32_t percentFinished,
                                    void* userData);
```

Argument	Description
`percentFinished`	0 - 100, inclusive. Use this for a progress bar or similar.
`userData`	The user data that was supplied to `REXCreate()`.

Return value	Description
`kREXCallback_Abort`	Abort creating the object. This causes `REXCreate()` to return `kREXError_OperationAbortedByUser`.
`kREXCallback_Continue`	Operation will continue.

The user defines this function if he wants to be able to show a progress bar or let the user abort creating a REX object. Creating a REX object could potentially take a substantial amount of time and is typically done directly after loading a file into RAM, so it is preferrable to present the whole load file/`REXCreate()` sequence as a ”Loading File…” operation, to the user.

The pointer to the function is supplied as an argument to `REXCreate()` if a callback is wanted. The callback function will then be called regularly during the create operation. The `userData` argument will hold the user-supplied data given to `REXCreate()`. There are no restrictions on the user data; you can do whatever you want with it.

Example:

```
REXCallbackResult MyCallback(REX_int32_t percentFinished, void* userData)
{
     char title[40 + kFileNameSize + 1];
     sprintf(title, ”Reading file: %s\n”, userData);
     SetProgressBarTitle(title);
     SetProgressBarPercent(percentFinished);

     if (UserClickedCancelButton()) {
          return kREXCallback_Abort;
     } else {
          return kREXCallback_Continue;
     }
}


// Somewhere else in the program
…
REXError error = REXCreate(&rexObject,
                           fileBuffer,
                           fileSize,
                           MyCallback,
                           fileName);
…
```

3.6 Preview Playing

The preview play API is intended to be used in file open dialogs etc, to preview REX2 files before they are loaded and used in the application. The REX DLL must have been loaded using `REXInitializeDLL()` and a REX object must have been created using `REXCreate()` for these functions to be available.

One of these functions may be called from interrupt level code: `REXRenderPreviewBatch()`.

3.6.1 REXStartPreview

Starts preview-playing a REX object.

```
REXError  REXStartPreview(REXHandle handle);
```

Argument	Description
`Handle`	Handle of the REX object to be previewed.

Return value	Description
`kREXError_NoError`	Success!
`kREXError_OutOfMemory`	Preview cannot be started because of low memory.
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidHandle`	The handle does not refer to a REX object currently in memory.
`kREXImplError_IsBeingPreviewed`	This REX object is already being previewed.

The file to be previewed must have been loaded into memory in its entirety, and a REX object must have been created from the file, using `REXCreate()`.

It is not possible to preview the same file more than once at a time. However, it *is* possible to preview multiple files at once, if you would ever want to do such a thing.

Having started preview-play, you should call `REXRenderBatch()` from your sound-card callback or interrupt routine. The play position will advance for each time you call `REXRenderBatch()`, and the preview-played file will play back looped until you stop playback by calling `REXStopPreview()`.

3.6.2 REXStopPreview

Stops preview playing of a REX object.

```
REXError  REXStopPreview(REXHandle handle);
```

Argument	Description
handle	Handle of a REX object currently being previewed.

Return value	Description
kREXError_NoError	Success!
kREXImplError_DLLNotInitialized	The REX DLL has not yet been initialized.
kREXImplError_InvalidHandle	The handle does not refer to a REX object currently in memory.
kREXImplError_NotBeingPreviewed	Preview-playing of this REX object has not yet been started.

It is not allowed to call this function on an object unless preview play of it has been started.

3.6.3 REXSetPreviewTempo

Sets the tempo of a REX object being previewed.

```
REXError  REXSetPreviewTempo(REXHandle handle,
                             REX_int32_t tempo);
```

Argument	Description
handle	Handle of a REX object being previewed.
tempo	The new tempo, in the format 123.456 BPM * 1000. To set the tempo to 135 BPM, set this parameter to 135000 etc. The range is 20 – 450 BPM, inclusive.

Return value	Description
kREXError_NoError	Success!
kREXImplError_DLLNotInitialized	The REX DLL has not yet been initialized.
kREXImplError_InvalidHandle	The handle does not refer to a REX object currently in memory. Maybe it has already been deleted.
kREXImplError_InvalidTempo	The given tempo is out of range.

This function may be called at any time, from the moment `REXCreate()` *returns* up to the moment *before* the call to `REXDelete()`, even though preview play has not been started. If called correctly, it will never fail.

3.6.4 REXRenderPreviewBatch

Renders the next N samples of the previewed REX object.

```
REXError  REXRenderPreviewBatch(REXHandle handle,
                                REX_int32_t framesToRender,
                                float* outputBuffers[2]);
```

Argument	Description
`handle`	Handle of a REX object being previewed.
`framesToRender`	The number of sample frames to render from the previewed REX object. The range of this parameter is 0 – 65536 sample frames, inclusive.
`outputBuffers`	Array of two float pointers to one or two buffers that will receive the rendered data for the left (`ouputBuffers[0]`) and right (`ouputBuffers[1]`) channels. The pointers may *not* point at the same buffer. If `outputBuffers[1]` is NULL, the output will be mixed to mono in the buffer pointed to by `outputBuffers[0]`. If two buffers are supplied and the previewed file is a mono file, the output will still be stereo. The output data ranges from –1.0 to 1.0, inclusive.

Return value	Description
`kREXError_NoError`	Success!
`kREXImplError_DLLNotInitialized`	The REX DLL has not yet been initialized.
`kREXImplError_InvalidHandle`	The handle does not refer to a REX object currently in memory.
`kREXImplError_InvalidArgument`	Something is wrong with one of the parameters. With the *Testing* version of the DLL, a debug string has been output to help you detect what the problem is.

The specified number of frames will be rendered in the buffer(s). You can call this function as many times as you wish. The previewed file will loop infinitely.

It is safe to call this function from a different thread than the rest of the functions in the API. The function is guaranteed not to allocate memory or do things that are not allowed at interrupt level. Please note that this *does not* mean that it is safe to do multiple simultaneous calls to `REXRenderPreviewBatch()`, each from a separate thread!

The only restrictions on when you can call this function on a REX object are quite obvious: you cannot call it unless you've first created an object with `REXCreate()`, and you should make sure you stop calling it *before* destroying the object with `REXDelete()`.

No sound will be output (the buffers are filled with 0.0) before you explicitly start the preview with `REXStartPreview()` and after you stop it with `REXStopPreview()`. After stopping the preview it will take some time (1280 frames at 44.1 kHz output sample rate, i.e. 29 ms) for the sound to *completely* decay into silence. If you wish to avoid pops and clicks you should continue rendering preview batches for this long after your call to `REXStopPreview()`.

The output data will be at the current output sample rate, which is initially the file's original sample rate, but can be set with `REXSetOutputSampleRate()`. Note that changing the output sample rate for a REX object will also change the sample rate of slices rendered with `REXRenderSlice()`.

3.7 Data Structures

3.7.1 REXInfo

General information about a REX object. Is retrieved by calling `REXGetInfo()` on an object.

```
typedef struct {
        REX_int32_t fChannels;
        REX_int32_t fSampleRate;
        REX_int32_t fSliceCount;
        REX_int32_t fTempo;
        REX_int32_t fPPQLength;
        REX_int32_t fTimeSignNom;
        REX_int32_t fTimeSignDenom;
        REX_int32_t fBitDepth;
} REXInfo;
```

Field	Description
`fChannels`	Number of channels. At the moment, this value always amounts to 1 or 2.
`fSampleRate`	Sample rate of the file's PCM data. 1.0 kHz – 1.0 MHz.
`fSliceCount`	Number of slices in the object. This value *may be zero*.
`fTempo`	Tempo set when exported from ReCycle, 123.456 BPM stored as 123456 etc.
`fOriginalTempo`	Original tempo of loop, as set up in ReCycle with the Left/Right locators and the Bars/Beats/Sign controls. In the same format as `fTempo`.
`fPPQLength`	Length of loop. The resolution is always 15360 PPQ (*Pulses Per Quarter-note*). This means that one bar in 4/4 time amounts to 15360*4*(4/4) = 61440 *Pulses*, one bar in 6/8 time is 15360*4*(6/8) = 46080 pulses, etc.
`fTimeSignNom`	Time signature nominator (the "6" in "6/8").
`fTimeSignDenom`	Time signature denominator (the "8" in "6/8").
`fBitDepth`	Original bit depth of ReCycle:d file. Please note that all processing in the REX DLL is carried out in 32-bit floating point format, meaning that regardless of the original resolution of the file you will obtain the best sound quality by using the floating point data directly, and not truncating or dithering it.

3.7.2 REXCreatorInfo

Optional information about a REX2 file, that the creator of the file may have added to it. The information can be retrieved by calling `REXGetCreatorInfo()` on a REX object. The strings are at most `kREXStringSize` characters long, *excluding the terminating NULL character*.

```
typedef struct {
     char fName[…];
     char fCopyright[…];
     char fURL[…];
     char fEmail[…];
     char fFreeText[…];
} REXCreatorInfo;
```

Field	Description
`fName`	The name of the REX2 file's creator.
`fCopyright`	Copyright information.
`fURL`	Where the creator can be found on the Internet.
`fEmail`	The E-mail address of the creator.
`fFreeText`	Any additional text that the creator may have added.

3.7.3 REXSliceInfo

Information about a slice in a REX object. Is retrieved by calling `REXGetSliceInfo()` on an object.

```
typedef struct {
       REX_int32_t fPPQPos;
       REX_int32_t fSampleLength;
} REXSliceInfo;
```

Field	Description
`fPPQPos`	Zero-based PPQ position of slice in loop. The resolution is 15360 PPQ. This position is always greater than or equal to zero and less than `fPPQLength`, which is found in `REXInfo`. Note that the first slice of a loop does not necessarily have to be at PPQ position zero.
`fSampleLength`	Length of rendered slice at the REX object's output sample rate.

4. APPENDIX A – FILE FORMAT ATTRIBUTES

The REX API lets you import and use three different file formats: REX-, REX2- and old-style ReCycle Document (".RCY") files. The API automatically detects and checks the validity of a file supplied to `REXCreate()`, but you will still have to know how to distinguish the files before you load them into memory and create the REX objects. Thus:

File Format	MS-DOS Extension	Mac OS File Type
REX2	".RX2"	"REX2"
Old-style REX	".REX"	"REX "
Old-style RCY	".RCY"	"RCSO"

The Mac OS creator code for ReCycle is "ReCy".

5. APPENDIX B – SYSTEM REQUIREMENTS

5.1 Windows

The REX DLL requires Windows 7 or later.

5.2 Mac OS X

The REX framework requires OSX 10.7 or later.
The REX framework includes a Universal 2 binary supporting both Intel(x86-64) and Apple Silicon(ARM64) architectures.

6. APPENDIX C – RECYCLE'S CLIPBOARD SUPPORT

Every application has its unique strengths and weaknesses. Sometimes when working with sound files the best solution is to use several applications in tandem, in order to get the job done as quickly and easy as possible. When designing ReCycle 2.0 we were sometimes tempted to bring every feature conceivable into the program and make it a full-fledged audio editor, but we didn't do that. Instead we decided to focus ReCycle on just what it does best: slicing audio files. Still, we realized that people working with ReCycle will sometimes need more features and therefore be bringing files back and forth between applications. Saving temporary files is a hassle. How can we make ReCycle integrate more smoothly with other audio applications? Enter the clipboard.

6.1 Copy Loop

The standard Ctrl/Command+C copy menu item in ReCycle bears the somewhat curious name "Copy Loop". Instead of the usual copy behavior, which is to copy the current selected stuff onto the clipboard, this command puts the current document on the clipboard, regardless of what is selected, in three different formats simultaneously:

1. A *MIDI file*, just like the one that would be generated if the user exported the loop as a MIDI file. Useful to bring MIDI data into a sequencer program when working with a software sampler.

2. A *WAVE file*, with just the unprocessed data between the Left/Right locators in ReCycle. This is maybe not so useful when working with other applications, but more so inside of ReCycle itself - to extract portions of a large file for individual ReCycle:ing.

3. A *REX2 file*, just like the one that is saved to disk. This is the most useful format when working with other applications. User just presses CTRL+C in ReCycle and then pastes the ReCycle:d file onto an audio track in the sequencer or into a sampler's keymap.

6.2 Paste as New Document

Akin to the Copy Loop command, in ReCycle the standard Ctr/Command+P paste command has been renamed "Paste as New Document" to better describe what it does. Instead of pasting data at the current cursor position, ReCycle creates a whole new document from data that is ”pasted” into the program.

”Paste as New Document” is enabled whenever there's a complete WAVE RIFF (”.WAV”) file available on the clipboard. The clipboard file has the same constraints as a file loaded into ReCycle from disk - if any of these constraints are not met ReCycle will inform the user about the problem.

6.3 Clipboard Formats

The format names/flavors that ReCycle uses to identify data on the clipboard differ a bit between Windows and Mac, simply because this is very platform-dependent technology. On Windows we use the standard clipboard APIs while on Mac OS X we use the Scrap Manager APIs. In short, for the respective formats, this is what we use:

Format	Windows Clipboard Format	Mac OS Scrap Manager Flavor
MIDI	”SMUGGLER_MIDI”	”MIDI”
WAVE	CF_WAVE	”WAVE”
REX2	”SMUGGLER_REX2”	”REX2”

7. APPENDIX D – REX COMPLIANCE TEST PROTOCOL

This section describes the ordeal that your program has to go through before you can label it a true REX application. The test won't take many minutes to do and you'll get a lot in return. All files that you need are available in the "Test Files" folder, which is installed with the REX SDK.

7.1 Loading Files

There are a couple of different situations when files must to be treated differently when they are loaded: some loops that your application should *not* be able to load, but instead respond with a proper error message, and some that are possible to load but require extra attention.

Test these loops in every way the application supports loading ReCycle files: using the standard "Open" dialog, as the result of a Drag & Drop action, from the "Recent Files" part of the "File" menu, or by using "Paste" on the "Edit" menu, if the application supports that. This table describes how the files should be treated:

File	Expected Behavior
120AllMuted.rx2	Error message: *"This ReCycle file does not contain any active slices."* All slices between the left and right locators are muted so there's nothing to play back.
ErrorCorrupt.rx2	Error message: *"The format of this file is unknown or the file is corrupt."*
ErrorCorrupt2.rx2	Error message: *"The format of this file is unknown or the file is corrupt."*
ErrorTooNew.rx2	Error message: *"This REX2 file was created by a later version of ReCycle and cannot be loaded with this version of the REX DLL. Please update the DLL, then try again."*
ErrorLoopLengthNotSet.rcy	Error message: *"This ReCycle file cannot be used because its 'Bars' and 'Beats' settings have not been set."* Means that the length of the loop has not been specified so the original tempo cannot be determined.
120RcyTest.rcy	This is a file in ReCycle 1.6/1.7 format. Test that your application accepts the file type.
120RexTest.rex	This is a file in REX format (not REX2!). Test that your application accepts the file type.

7.2 Import and Playback

The remaining test files have been designed to make sure all features of a REX file are correctly implemented in your application. For each file there is a corresponding WAVE file that shows how the file is supposed to sound when it has been imported into your program. Some of the wave files are two bars to show how the files are supposed to loop, while files for testing other properties are just one bar.

Open each of the following files in your application. Set the tempo and time signature given in the table and play a one bar loop. Then play the corresponding wave file and make sure they sound exactly the same. Note that the tempo in the table refers to the number of *beats* per minute, not *quarter notes* per minute, which is common to use in many software packages (if that's the case you'll have to halve the tempo for the 7/8 loop).

Tip: for sequencers like Cubase and Logic it may be easiest to listen by importing the REX and wave files simultaneously to two different tracks and align the start of the two files.

File	Tempo	Sign	Length	Description
120Mono.rx2	120	4/4	1.0	120Stereo.rx2, but in mono (mixed left and right).
120Mono24Bit.rx2	120	4/4	1.0	Same as above but with better resolution.
120Stereo.rx2	120	4/4	1.0	Test stereo. Slices are panned hard left and right.
100WeirdSampleRate.rx2	100	4/4	1.0	Odd sample rate of 15.687 kHz.

File	Tempo	Sign	Length	Description
120TransmitAsOneSlice.rx2	120	4/4	1.0	This loop has only one slice. Thus its tempo should not change with your applications tempo.
120FourBeats.rx2	120	4/4	0.4	Four beats. Wave file is two bars.
120SevenEights.rx2	120	7/8	1.0	Seven eighths. Wave file is two bars.
120ThreeBeats.rx2	120	4/4	0.3	Three beats. Wave file is two bars.
120Gated.rx2	120	4/4	1.0	Same as 120FourBeats except for the gate setting (cuts off the end of slices.) If this doesn't work, your application probably stacks slices after each other rather than using the proper PPQ positions. Wave file is two bars.
120GatedMuted.rx2	120	4/4	1.0	As 120Gated.rx2 but some of the slices are muted. Make sure the remaining slices are played at the correct position, e.g. the first slice is mute so the loop should not start playing directly. Wave file is two bars.
240FiveHundredSlices.rx2	240	4/4	10.0	A loop with 500 slices. Your application may or may not support that many slices. Instead, you could choose to open the maximum amount of slices your application can handle or simply refuse to open the file. If the former, make sure the loaded slices play at the correct tempo.
450OneHundredBars.rx2	450	2/4	100.0	Long loop with two slices. Two sines, the first plays from start and the second at 50 Bars. Make sure the imported slices appear at the correct positions.

7.3 Preview Play

If your application supports previewing of REX files, for example in the file open dialog, test all of the loops above with your preview player code. Make sure that the files play as expected (simple to compare with the wave files if the preview player supports both formats!) and that any information presented about the files is correct.