

Spring E. Thing Presents: SkinTunes (Version 1)

Contents

What is SkinTunes?	2
How to Implement SkinTunes	3
Prerequisites	3
The Short Answer (For Pros)	3
The Long Answer (For Newbies)	4
Supported Tunes.....	10
API Documentation	11

Spring E. Thing Presents: SkinTunes (Version 1)

What is SkinTunes?

SkinTunes is a standalone Lua script addon for Dr. Robotnik's Ring Racers (DRRR) that enables any racer of DRRR to have unique music tracks contextual to one such racer. The script is intended to be loaded before any racers that use the script. Though such load ordering is unnecessary on a technical level, a majority of racers will become non-functional with SkinTunes on a practical level. A 'SkinTune', not to be confused with the 'SkinTunes' script, is a music track present in the unmodified DRRR game that has been uniquely adapted into SkinTunes to allow racers using SkinTunes to swap in their own songs over those music tracks. If one finds themselves curious as to how they can take advantage of SkinTunes in their own racers or racer packs, continue to the sequential sections of this README.

Spring E. Thing Presents: SkinTunes (Version 1)

How to Implement SkinTunes

Prerequisites

To engage with SkinTunes, there are few prerequisites, all of which one would be expected to have met regardless if one has interest in SkinTunes in the first place:

- An existing PK3/WAD of a DRRR racer or racker pack
- Basic operational knowledge of the PC software 'SLADE'
- A download of DL_SkinTunes_V<X>.lua in which the <X> matches the versioning of this README

This is all that is required of one's self to follow the rest of the implementation guide.

The Short Answer (For Pros)

1. In SLADE, create an audio lump for the SkinTune one wishes to interact with. Create a MUSICDEF lump that signals to DRRR that the audio lump is a music track by writing 'Lump <LUMPNAME>' in the lump.
2. Then, create a Lua lump in a place within the WAD/PK3 addon that will execute automatically upon the addon being loaded in DRRR. This will be the lump that connects the audio lumps from before to SkinTunes.
3. Now, begin by writing a nil check for the SpringESkinTunes library. Use this to abort the script if the library is nil as this means SkinTunes isn't loaded to interact with.
4. Next, use SpringESkinTunes.Set(string skinName, string tuneInfoKey, string/table tuneSong) to register the audio lumps created earlier with SkinTunes. The first parameter is the S_SKIN name value of the racer this call affects, the second argument is the unmodified DRRR music track this call effects, and the third argument is the audio lump name(s) this music track should play for a song instead when one plays as the skinName racer. For example, if one were to desire to set the invincibility song of Sonic to an audio lump named SONIIN, the call would be SpringESkinTunes.Set("sonic", "invinc", "SONIIN"). For more detailed information about this call, check the 'API Documentation' section of this README.
5. Finally, save any changes to this WAD/PK3 and load both it and the SkinTunes standalone Lua script as addons in DRRR making certain to load SkinTunes *before* the WAD/PK3. If all went to plan, one just made their first SkinTune! Enjoy!

Spring E. Thing Presents: SkinTunes (Version 1)

The Long Answer (For Newbies)

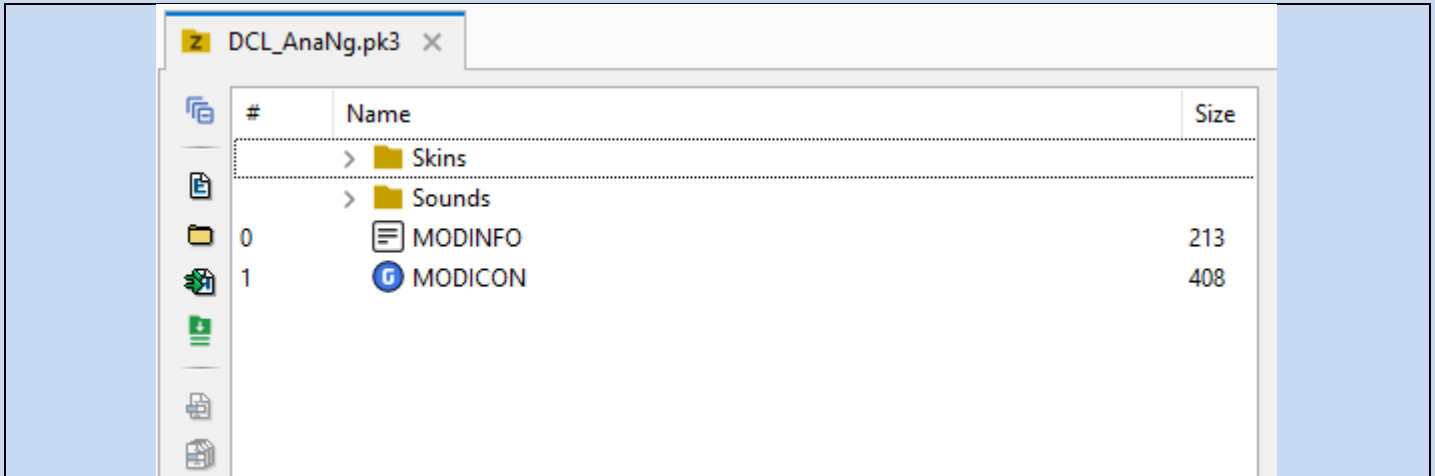


Figure 1

Figure 1 depicts SLADE as an open program with a PK3 loaded into it. This PK3 is the one that will be operated upon for the majority of this implementation answer. SkinTunes will work just as well in cooperation with a WAD and the prerequisites for implementation will be assumption enough that one can adapt this answer for a WAD instead of a PK3. In addition, there's no reason one cannot apply this answer to a file that represents a pack of racers instead of just one individual racer. For example's sake, the goal of this answer has been determined to be to allow the fictitious 'Ana Ng' racer to use a unique song when under the effects of the 'Grow' power-up. As one will notice perhaps more clearly later on, the PK3 file is named 'DCL_AnaNg.pk3' to reflect this.

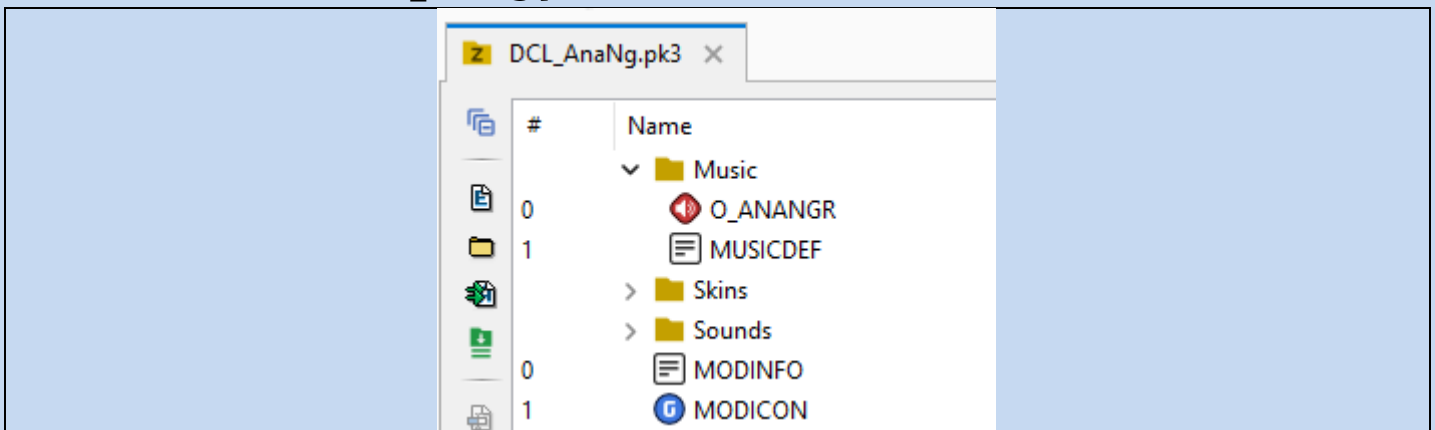


Figure 2

As figure 2 depicts, one should first create an audio lump to represent the song that will replace the usual song of the 'Grow' DRRR music track. The audio lump should go in the 'Music' directory in the root of the PK3. The data of the lump is recommended to be a 44.1KHz Stereo OGG audio file as this has the best compatibility with the DRRR engine but other options are available for those willing to fiddle about with that sort of thing. The name of the lump can be anything up to 8 characters long, however it must begin with 'O_' for the sake of the DRRR engine to recognize it as a song in-effect giving one only up to 6 characters of name space to work with. Since fundamentally to the DRRR engine this lump name must be unique from any other song in DRRR and furthermore any other SkinTune

Spring E. Thing Presents: SkinTunes (Version 1)

song, it is recommended that the chosen 6 character name begin with 4 characters matching the skin lump prefix of the affected racer followed by 2 characters to uniquely identify this lump beyond that of one's choosing. This is a song for Ana Ng (who shall be assumed to have the prefix ANNG) to replace the song that plays for the 'Grow' music track, so as figure 2 depicts the lump has been named O_ANANGR.

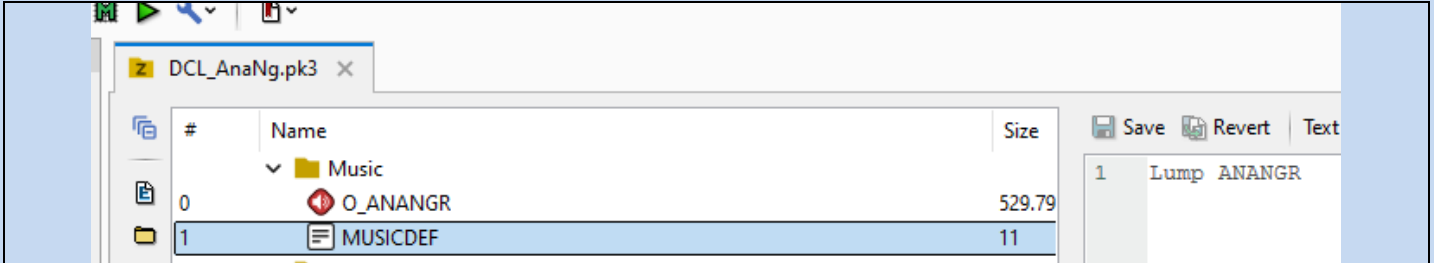


Figure 3

Figure 3 depicts a MUSICDEF lump one should now create. It should be a sibling in the PK3 heirarchy to the audio lump created moments ago. Although this lump is usually utilized to fully register new music tracks to DRRR, the lower level operation of SkinTunes will only require the first line of a music track registry to be entered per audio lump. Simply write 'Lump ANANGR' to match that of the audio lump without the O_ prefix and the MUSICDEF is complete.

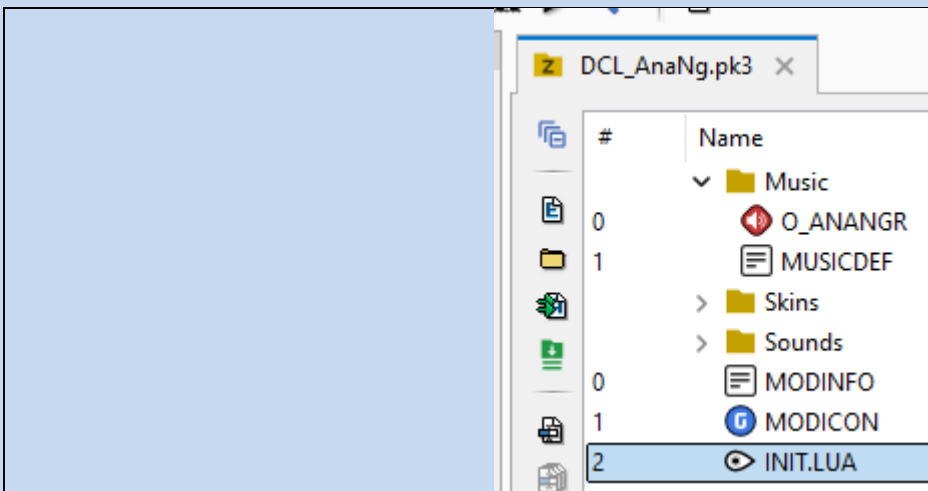
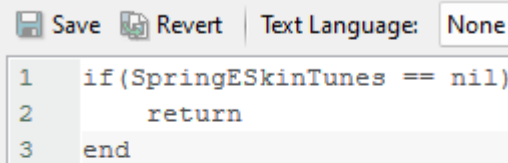


Figure 4

Figure 4 depicts a Lua lump created in the root of the PK3 named INIT.LUA that one can choose to create now if they so wish. There are several ways to get a Lua lump within a PK3/WAD to automatically execute upon the PK3/WAD it is a part of being loaded in the DRRR engine, INIT.LUA is just one such method, so please go with one's own preference for where to put the Lua lump and what it should be named should they know better. So long as one is confident that their methodology will cause the Lua lump they create to execute automatically upon loading the PK3/WAD, there will be no issue.

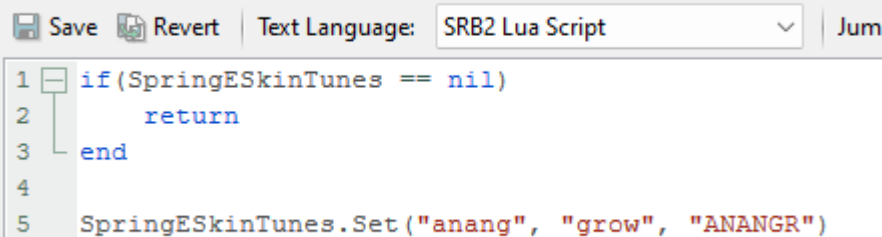
Spring E. Thing Presents: SkinTunes (Version 1)



```
1 if(SpringESkinTunes == nil)
2   return
3 end
```

Figure 5

Figure 5 depicts how one should begin the contents of the Lua lump just created. The entire API for SkinTunes is contained within the SpringESkinTunes namespace. This namespace only exists when SkinTunes is loaded and trying to access it if SkinTunes isn't loaded will lead to runtime errors, so simply begin the lump with a nil-check to abort the lump's execution should the namespace be missing.



```
1 if(SpringESkinTunes == nil)
2   return
3 end
4
5 SpringESkinTunes.Set("anang", "grow", "ANANGR")
```

Figure 6

Figure 6 depicts how one can finally engage their prior created lumps with SkinTunes. The SpringESkinTunes.Set() method allows one to assign any number of songs to supported music tracks filtered by the racer it represents. This is referred to as "registering" with SkinTunes. The first argument should be equal to the S_SKIN name value of the racer affected, the second argument should be equal to one of the supported music tracks found in the 'Supported Tunes' section of this README, and the third argument should be a string or a table of strings with each string being a different audio lump name that has been registered through MUSICDEF. As demonstrated by figure 6, this couldn't be simpler in this example. Ana Ng's S_SKIN name let's pretend is anang, the 'Grow' music track's supported tune name is 'grow', and the song the example desires to play is the same as the name we used for our music audio lump in MUSICDEF, that being ANANGR. Technically, at this point if all one desires is to engage with SkinTunes at the most basic level, one is done and dusted. However, instead of progressing to the end of the answer immediately, this moment will first be taken to highlight some other core parts of the SkinTunes API for more delicate control over interaction with SkinTunes. Let it be known and clear that no information in the rest of this answer prior to figure <X> will actually be used in the PK3 being worked on, all for example's sake.

Spring E. Thing Presents: SkinTunes (Version 1)

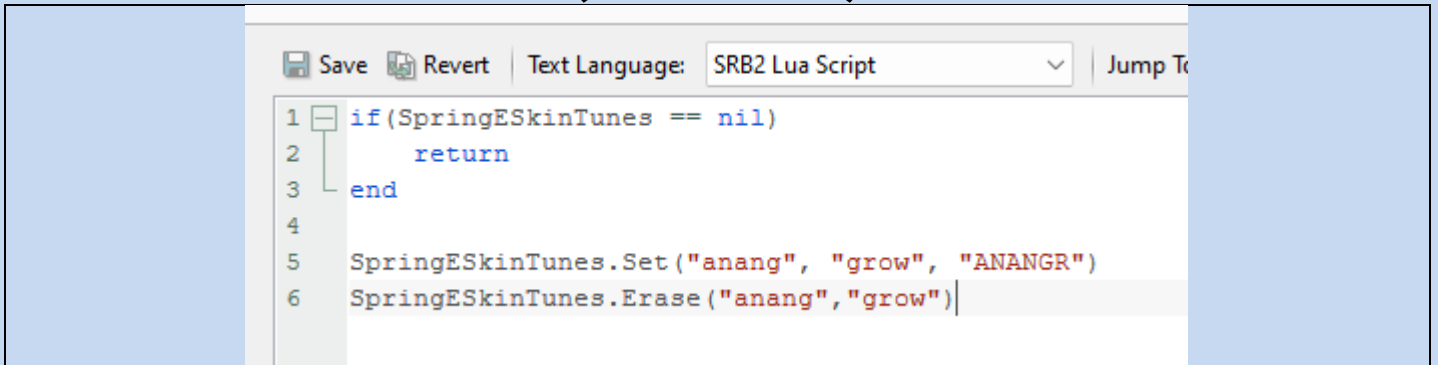


Figure 7

Figure 7 depicts the `SpringESkinTunes.Erase()` method. This method is very powerful if used properly but by-proxy volatile if used carelessly. The core function of the method is self-explanatory, if `SpringESkinTunes.Set()` registers, then `SpringESkinTunes.Erase()` de-registers, that is to say it removes information from SkinTunes. Each successive argument to this method call applies a tighter level of precision to exactly what is removed from SkinTune's memory. If no arguments are provided, then everything in SkinTune's memory will be wiped, even register actions from other addons interacting with SkinTunes! If only a racer `S_SKIN` name is provided as the first argument, then just that racer will be forgotten by SkinTunes. If a racer `S_SKIN` name is provided first followed by a supported music track name second, then just the information related to that track for that racer will be lost. This would be elaborated further, but one should surely be able to extract their own understanding from there as to what successive arguments in the method call do.

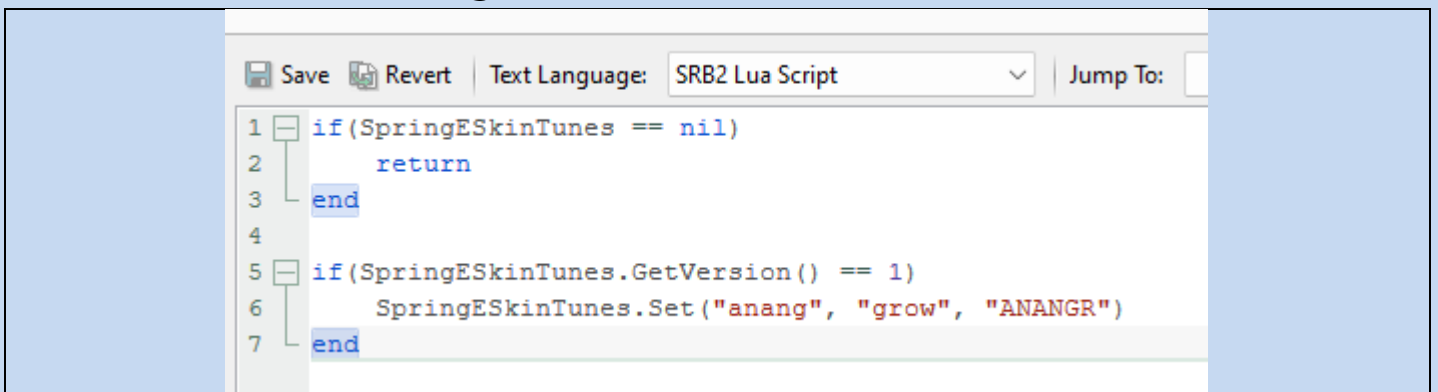


Figure 8

Figure 8 depicts the `SpringESkinTunes.GetVersion()` method. Though not of much use as of yet, this method can be used even now to future-proof interactions with SkinTunes. By conditioning calls to methods like `SpringESkinTunes.Set()` and `SpringESkinTunes.Erase()` based upon `SpringESkinTunes.GetVersion()`, those who interact with SkinTunes can account for a future where something drastic changes about SkinTunes and all of a sudden those calls to other API methods that would now throw warnings and errors at the user instead silently fail or fall back to an alternative.

Spring E. Thing Presents: SkinTunes (Version 1)

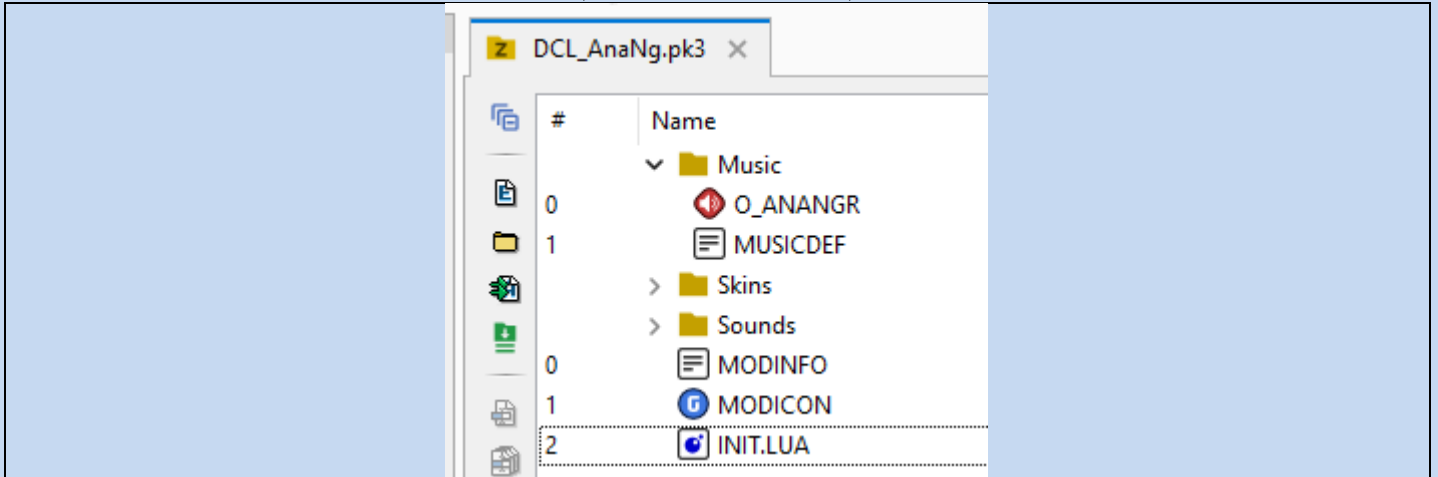


Figure 9

Figure 9, much like the earlier figure 1, depicts the SLADE program open on the PK3 one has been operating on this whole time. That is because one is now complete in interacting with SkinTunes and is ready to test it out in DRRR! First, save the PK3. Then, close down SLADE as it won't be needed anymore. Next, put the PK3 in the 'addons' subdirectory of the DRRR install folder followed by the SkinTunes Lua file downloaded as a prerequisite. It doesn't matter if these files share a subdirectory within the 'addons' directory or not, just so long as both can be found by DRRR when choosing to load addons however one likes to load their addons.

silently fail or fall back to an alternative.

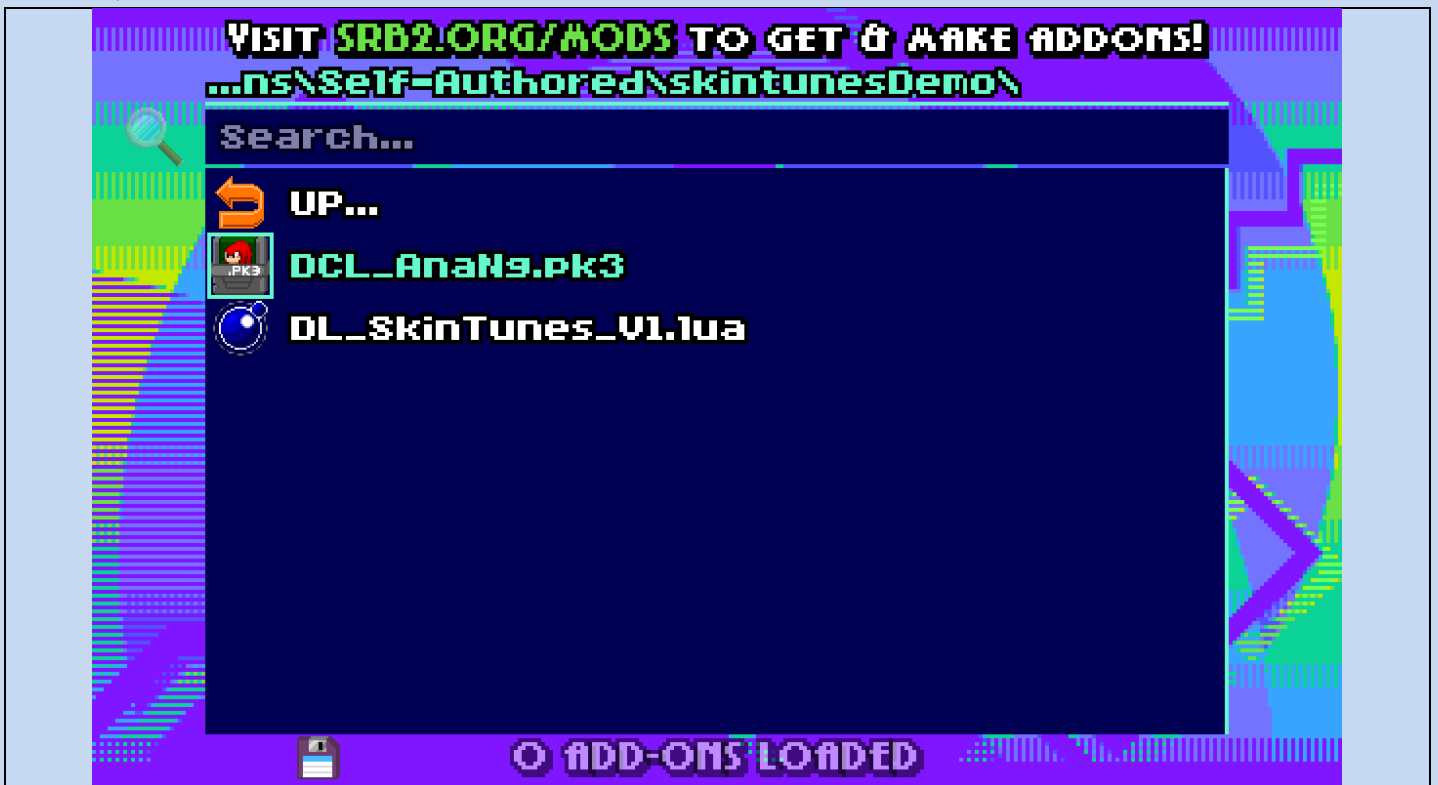


Figure 10

Figure 10 depicts DRRR, now open and on the 'Addons' menu. When activating addons that interact with SkinTunes, it is crucial that SkinTunes is activated first. This is, of course, first

Spring E. Thing Presents: SkinTunes (Version 1)

because one wrote that nil check earlier that will make the Lua script in the affected racer addon's lumps completely ineffective if one doesn't do this, but also if the nil check were to be removed it would only result in errors from loading the racer addon before SkinTunes was ready to be interacted with.

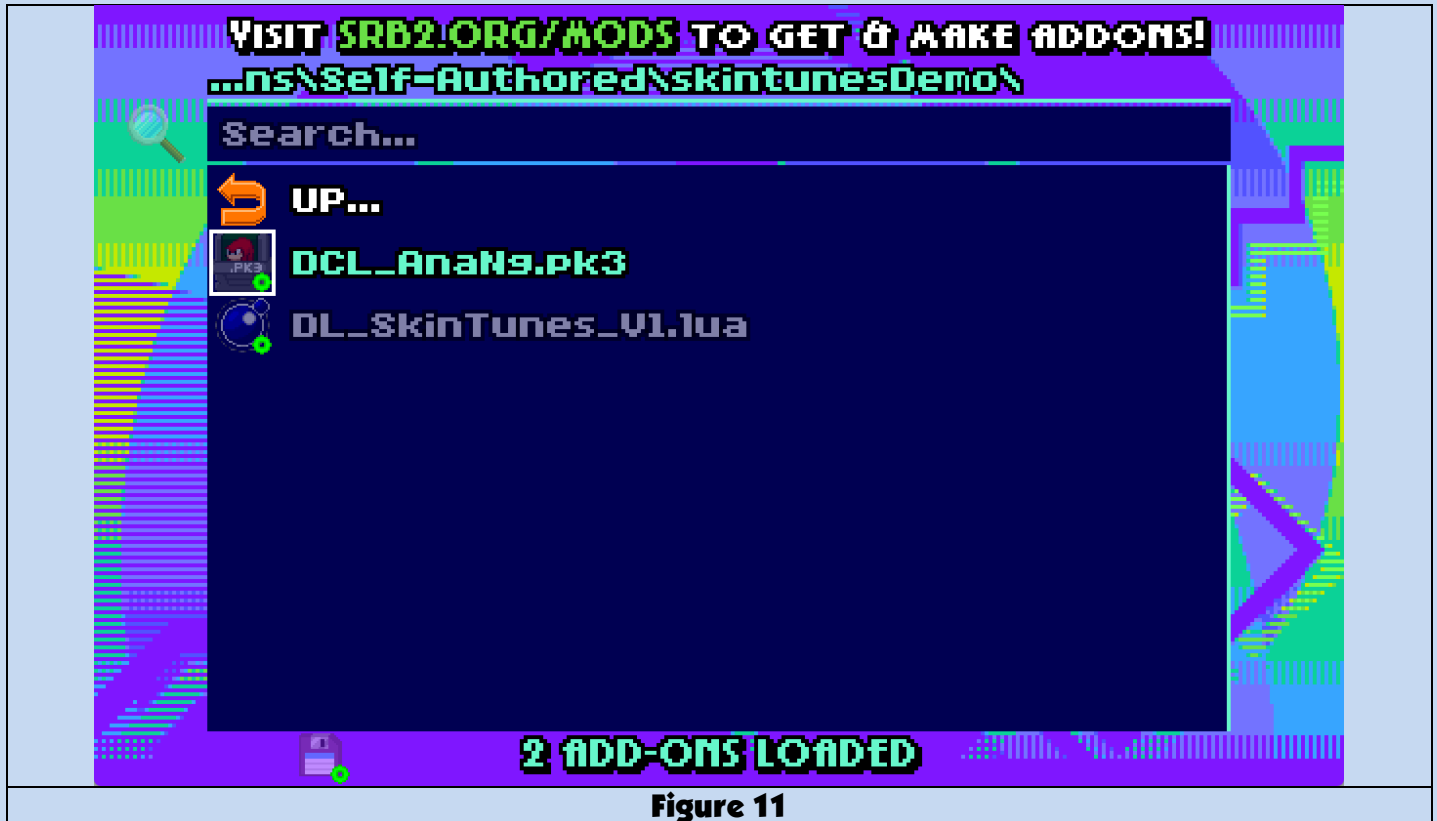


Figure 11

Figure 11 depicts both the SkinTunes addon activated and the addon that interfaces with SkinTunes activated, the former activated before the latter. If one is at this point, one is now ready to fully enjoy SkinTunes with the affected racer!

Spring E. Thing Presents: SkinTunes (Version 1)

Supported Tunes

Tune Name	Playback Scenario
invinc	Racer is under the effects of invincibility
grow	Racer is under the effects of grow

Spring E. Thing Presents: SkinTunes (Version 1)

API Documentation

Method	Description	Arguments		Return Value	
Set (string skinName, string tuneInfoKey, string/table tuneSongs)	Associate a racer with the concept of altering an unmodified DRRR music track to play a new song or selection of songs.	skinName	Who, according to S_SKIN name value, is getting associated with SkinTunes?	boolean	Will be 'true' only when every detail of the arguments in the call were compatible with SkinTunes.
		tuneInfoKey	What music track in unmodified DRRR are they interacting with?		
		tuneSongs	What audio lump(s) with a MUSICDEF entry should the tune represented by tuneInfoKey play instead?		
Erase (string skinKey, string tuneInfoKey, string/table tuneSongs)	Disassociate increasingly precise information from SkinTunes. All arguments are optional, the more one provides then the more one can erase. One is advised to practice caution with this method as leaving all arguments excluded will wipe all the SkinTune entries, even ones from other addons!	skinKey	Filter to only erase a specific skin by its S_SKIN name value.	table	A table with two indexed keys. The value at key 1 is a boolean which is only true when any degree of success was had in removing the provided arguments from SkinTunes. The value at key 2 is an integer equal to how many elements were removed. If no tuneSongs argument is provided, this will always equal an integer cast of the value at key 1. Otherwise, it will be how many items from tuneSongs were able to be successfully removed from SkinTunes.
		tuneInfoKey	Filter to only erase a particular tune that belongs to the skin associated with skinKey.		
		tuneSongs	Filter to only erase particular song(s) associated with the tuneInfoKey key.		
GetVersion ()	Read what version of SkinTunes is loaded. Intended to adjust implementation responses accordingly.	N/A		number	1