# Help for Sonic R Mod Loader 1.0.0

InvisibleUp

July 23, 2015

Welcome to the documentation for the Sonic R Mod Loader. This document covers 4 main areas:

**Using the Mod Loader**
This section deals with how to operate the mod loader, installing and uninstalling mods. If you have no interest in creating mods of your own this is all you will need to read.

**How the Mod Loader Works**
This section describes the basic functionality of the mod loader. It explains how mods are stored and installed. Read this if you're interested in making mods, or are just curious as to how the program works.

**Creating Mods**
This section describes how to create a mod and the files that comprise a mod. This section assumes you have knowledge of programming in x86 assembly and are comfortable with editing text files. (If you don't know x86 ASM, read this anyways, but skip over the "Working with the EXE" section. You can still create some pretty cool mods without knowing how to program.)

**Adding Support for Additional Games**
This section describes how to extend the functionality of the Sonic R Mod Loader to support other games. This section assumes you either have access to or are comfortable creating a disassembly of the game you wish to add.

# 1 Using the Mod Loader

Note: The documentation for this section is written assuming you're using the Windows GUI. If you're using an interface for another operating system or a command line interface, refer to the documentation that came with that version of the program for specific details on how to operate it.

## 1.1 System Requirements

The Sonic R Mod Loader has very, very modest system requirements. In theroy if you got the DLLs to cooperate you could easily get it running on any Windows 95 or NT machine capable of running Sonic R. The program has been confirmed to work out of the box with the following system:

**Operating System** Windows 2000 or greater

**CPU** Pentium processor or greater

**RAM** 32MB or greater

**Disk** 1MB free space plus at least 10MB for mods.

**Display** VGA display (640x480x16)

**Shared Libraries**

- Microsoft Visual C++ Runtime (msvcr100.dll)
- SQLite 3 (sqlite3.dll)
- Jansson 2.5 or greater (jansson.dll)

Saying the system requirements are modest is an understatement, to say the least. With the exception of the Visual C++ library, which comes with just about everything, all the DLL files you will need are included.

## 1.2 Creating a Profile

(Screenshot of profile view)

Profiles are a simple way of keeping different game installations from mingling. You will need to create a profile when you start the program for the first time. Click the (+) button to add a new profile.

(Screenshot of profile wizard or whatever)

To start, you will need to choose the game you wish to mod. Some games have more than one revision, so make sure you select the right one.

After that, you will need to set the installation path of your game.

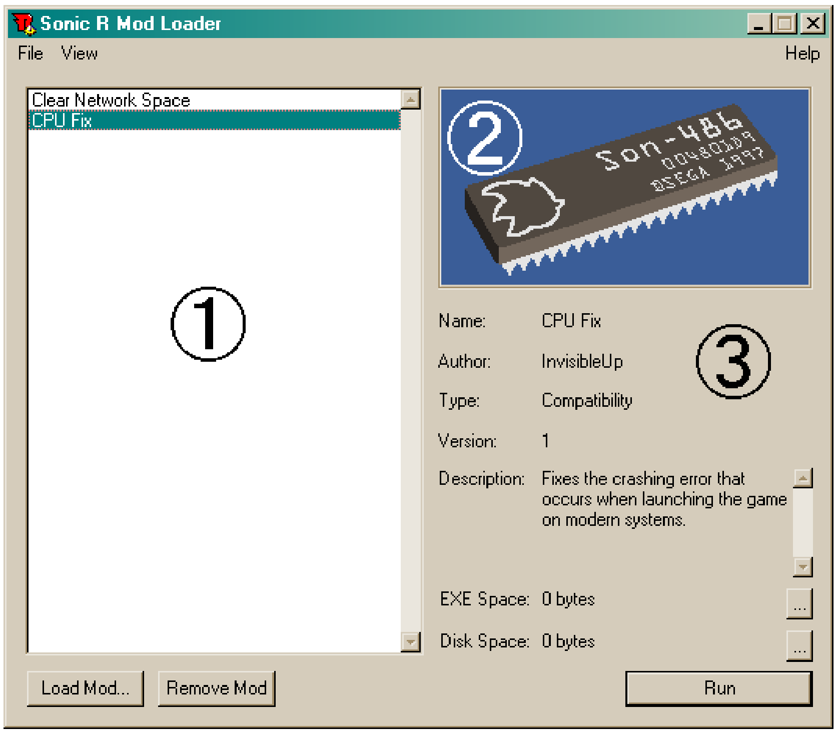
The mod loader will need write permissions to the game, so it will not work with programs installed in the "Program Files" folder unless you manually change it's permissions.

If you have selected a valid installation for the game, the loader will prompt you for a name. Enter a descriptive name for the profile.

Once that is complete, the profile selection screen will reappear. Select the profile you wish to run and click "Load".

## 1.3 The Mod Loader Interface

Once the process is complete, a screen should appear showing a list of installed mods. This is the where most actions within the program will take place.



Name: CPU Fix
Author: InvisibleUp
Type: Compatibility
Version: 1
Description: Fixes the crashing error that occurs when launching the game on modern systems.
EXE Space: 0 bytes
Disk Space: 0 bytes

Figure 1: The primary interface of the Sonic R Mod Loader

**1. List**

If a profile has just been created, the list will be blank. You will have to install some mods to fill the list. Clicking on an item in the list will change the Preview and the Details panes to show information about that mod. The “Remove Mod...” button will also remove whicher mod is selected.

2. **Preview**
   This is an optional picture provided by the mod’s developer meant to symbolize the mod. It is usually a picture of the content added, if it is a level, a character, or something of the sort. Sometimes, like this example, no content is being added, so an icon is added to quickly explain the purpose of the mod.

3. **Details**
   This is a list of some details about who made the mod and what the mod does. The following things will be listed:

   **Name**
   The name of the mod. It should be short, descriptive, and telling of what the mod offers.

   **Author**
   The name of the person or team who created the mod. It can either be the username of the creater or it can be the creator’s real name. If you’re having a trouble with a mod, ask the mod’s author, not the developer of this program.

   **Category**
   This is the role that the mod takes on. A list of valid category types can be found at 3.1.7.

   **Version**
   A brief summary of what a mod is supposed to do. Useful for if you forget a mod’s puropse.

   **Description**
   The version of the mod you have installed. Always a whole number. Allows you to tell at a glance if a mod needs updating. Make sure your mods are up to date!

   **EXE Space**
   The amount of space freed or used by the mod in the game’s main executable file. If you run out of space, use this to determine what to dump.

   **Disk Space**
   The amount of space used or freed by the mod in other files. This includes files the mod creates or deletes.

There are also 6 buttons that allow the user to preform various tasks. These buttons can also be found in the top menu bar, if that is preferred.

**Load Mod...**
Lets the user select a mod for installation. See 1.4 for information.

**Remove Mod**
Uninstalls the currently selected mod. See 1.8 for more information.

**EXE Space (...)**
Shows a view of all the spaces that were added or removed by the mod. See 1.6 to learn how to use the interface and 2.2 to learn what the interface is showing you.

**Disk Space (...)**
Shows a view of what files were changed and how much space was used in doing so. See 1.7 for details.

**Configure...**
Shows a list of all public variables that the user can customize. See 1.5 for more details.

**Play...**
Runs the game's main executable. Make sure that if you need a program like DXWnd to make the game work properly that you run that before you press this button.

## 1.4 Installing Mods

Installing a mod is a relatively simple process. Make sure the desired mod is already downloaded. The mod loader does not support directly downloading mods over the internet.

⚠ Make absolutely certain that the mod came from a trusted source. Because the mods typically modify executable files, there is nothing preventing a malicious mod maker from sneaking a virus into a mod. If in doubt, run a virus scanner on the game's installation directory before running it.

To install a mod, click on the button labeled "Load Mod...". Alternatively the mod may also be installed from the "Load Mod..." option in the "File" menu or by dragging and dropping the mod onto the mod list.

At this point a file browser window will appear asking for the location of the mod to install. Find and select the mod that is to be installed. The mods will be stored in a zipped folder (.ZIP), or for smaller mods, a JSON file.

There is a chance that a message will appear saying that other mods are required. If this message appears, you will not be able to install the mod until the dependencies are found and installed. There is no mechanism built into the mod loader to handle this automatically. The names and authors of the mods are provided so you can search for them on the Internet.

If the selected mod is already installed, the loader will alert you and installation will stop. If a newer version of an already installed mod is selected, the mod loader will ask the user if it is okay to upgrade the mod. Answering yes will cause the old version to be uninstalled and the new version to be installed in it's place. Selecting no will stop the installation.

At this point the installation should start. You will not need to take any action during the installation progress.

If, during the installation, the file to be patched runs out of free space, the installation will stop and any parts of the mod that have been installed will be installed. If this occurs, you will need to install a special type of mod designed to free up some space. See 2.2 for details.

## 1.5 Configuring Mods

Mods have the ability to set "variables", some of which can be set by the user. (See 2.4 for details on how exactly they work.) Variables that can be set by the user are known as "public varibles". These are useful for options that would be too difficult or obscure to be worth placing in the program itself.

Public variables can take on 3 forms.

**Numeric**

These are numbers. They have a minimum and a maximum range. Note that these ranges may be very high or very low, so if a large number is needed it is best to input it with the keyboard. Examples of such would be menu transition speed, character jump height, density of rain, etc.

**Dropdown List**

These are multiple options for an object or function in a mod that can chosen from. These should be options that would be not be well represented as numbers or as checkboxes. Examples would include menu color, character outfit, time of day, etc.

**Checkbox**

> These are values can be toggled on or off. Examples of such would be an option to always randomize courses, a switch to display dust behind a character, a rain toggle, etc. These should not be used to enable or disable a mod.

All variables are always enabled and displayed indepentendly, but keep in mind a mod developer could make it so that some public variables are ignored depending on the value of a dropdown list or checkbox.

Keep in mind that some public varaibles require the mod to be reinstalled after they are changed. This will be done automatically, but if there is no free space the variable will not be able to be changed until more space is cleared.

## 1.6 Viewing Patches

## 1.7 Viewing Modified Files

## 1.8 Removing Mods

Removing mods is a very simple process. Simply select the mod you wish to remove and press "Remove". The mod will then be removed.

If a mod depends on the mod you are trying to remove, the mod loader will ask you if you wish to remove that mod and continue. If you answer "No", the process will stop and your mod list will be unaltered. If you answered yes, both mods will be removed.

If, during removal, it is found that a mod required free space offered by the mod you are removing, removal will stop and your mod list will be unaltered. Add another space-removing mod or remove any mods that take up a lot of space if this happens.

# 2 How the Loader Works

This section describes the inner workings of the Sonic R Mod Loader. This will be useful reading if you're planning on makign a mod, working on the source of the mod loader or are just plain curious to see how it works.

## 2.1 How Mods are Stored

Mods are usually delivered to the user in a .ZIP archive. Inside the archive is a .JSON file defining the patches and other information needed to install the mod. Any other file in the archive is simply data that the mod uses during patches.

The format of the .JSON file is described in detail in section 3.

## 2.2 Introduction to Spaces

The mod loader is unique in that it operates much like your system's memory or hard disk works. It fits it's data wherever there is room, and nowhere else. Of course, there has to be a way for the mod loader to keep track of what data is being used. Spaces are the mod loader's way of dealing with this.

There are two types of spaces: Add and Clear. They act as you'd expect; Add spaces are sections in the file that are used by a mod or by the game itself, and Clear spaces are spaces that are not used by anything and are free for use by any mod that needs it.

Spaces are defined with a start, an end and a unique identifier. With the unique idenitifer, mods can refer to and use routines that another mod developer makes.

When a profile is first created some spaces are filled in corresponding to the already existing subroutines. This allows mod developers to either easily replace existing subroutines or to call existing subroutines during the mod's code

## 2.3 Patches

Patches are the commands issued to modify parts of a file or to change the status of spaces. Patches include the name of the file to modify (File), and a start byte (Start). Depending on the operation, it may also include an end byte (End), a unique identifier (UUID), the value of the bytes to add (Value), the format of the value (AddType) and the start and end of the destination (DestStart) and (DestEnd). There are several more values that can be added that will be described later.

### 2.3.1 Operations

There are 6 different operations a patch can preform:

**ADD**
: Searches for a clear space in the range of (Start) to (End) of the length of the patch's VALUE argument. If it is found, bytes are filled in starting from the beginning of the clear space. The used portion is turned into an add space with the UUID of (UUID), or an autogenerated one if (UUID) is not present, and the remainder is kept as clear. Aborts installation if there is no clear space.

**CLEAR**
: Creates a CLEAR space from the range (Start) to (End). Aborts if the space is occupied by another mod, but not if it is occupied by a space defined in the game definition.

**RESERVE**
: Searches for a clear space in the range of (Start) to (End) of the length (End) - (Start). If found, the space is marked as add with the UUID in (UUID) if present, or the old one if not, and no further action is taken. Fails on the same conditions that ADD fails. Mostly useful for reserving chunks in the special file dedicated for RAM for you mod's code.

**REPL**
: Applies a CLEAR operation from (Start) to the length of (Value), and then immediately applies an ADD operation on the newly cleared space. Fails on the same conditions CLEAR fails.

**COPY**
: Takes the bytes from (Start) to (End) and applies an ADD operation from (DestStart) to (DestEnd). Fails on the same conditions that ADD fails. . . .

**MOVE**
: Reads the bytes from (Start) to (End) into memory applies a CLEAR operation from (Start) to (End), applies a CLEAR operation from (DestStart) to (DestEnd), then applies an (ADD) operation from (DestStart) to (DestEnd) with the contents of memory. Fails on the same conditions that CLEAR fails.

These 6 operations should accomplish just about anything you would need to do. (In fact, I could get away with just 2, but that would be highly inconvienent for a lot of common tasks.)

Note that the mod loader does not support appending or deleting bytes from a file. The types of files the Sonic R Mod Loader was created to work on have very strict binary formats and will break spectacularly if the file size changes. Support for these operations may be added in the future, but it is not a priority.

### 2.3.2 Value Types

The (Value) argument can take on several forms, which changes where the bytes are retrieved from.

**Bytes**
Raw bytes in hexadecimal form. They are written exactly as given, after conversion from hex format. Use for very short snippets of data or code.

**Path**
A path to a file inside the mod's folder or archive. The data to be added is retrieved form the entire contents of the file. Use for resources or very large code samples.

**UUIDPointer**
The UUID of an add space, optionally followed by a (+ num) or a (- num). The data to be added is a 32-bit pointer corresponding with the space's (Start) value, which is than added or subtracted by the given value. This allows you to call subroutines or access memory variables located in other patches, and perhaps from other mods entirely.

**VarValue**
The UUID of a variable. The data to be added is the contents of the variable, in the datatype specified when the variable was first created. Use for calculations or code branching that is dependent on the status of other installed mods.

### 2.3.3 Cross-File Operations

The Sonic R Mod Loader is not just limited to operating on one file at a time. A mod developer can create patches that work on multiple files. This is useful for games that have their code or data split across many different files.

If a destination path (DestFile) is included and is different from the source file (File), the operations COPY and MOVE take their source from (Start) to (End) in the file (File) and output to (DestStart) to (DestEnd) in the file (DestFile).

### 2.3.4 Whole-File Operations

The Sonic R Mod Loader is best suited for altering small parts of a file, but it can also work with whole files. When the patch specifies (File) and/or (DestFile), but not (Start), (End), (DestStart) or (DestEnd), the 6 commands act slightly differently.

**ADD**
: Takes the contents of (Value) and creates a new file at (File) with those contents. An add space for that file is also created that is the size of that file. Aborts if the file already exists.

**CLEAR**
: Deletes the file at (Value), along with any associated spaces. Aborts if the space is occupied by another mod, but not if it is occupied by a space defined in the game definition.

**RESERVE**
: No operation.

**REPL**
: Applies a CLEAR operation on (File), then applies an ADD operation on (File).

**COPY**
: Applies an ADD operation on (DestFile), using the contents of (File) as (Value).

**MOVE**
: Reads contents of (File) into memory, applies a CLEAR operation of (File), and then applies an ADD operation on (DestFile) using the contents of memory as (Value).

## 2.4 Variables

Variables are bits of data defined by mods than can be altered by other mods or users. The game cannot modify or read these directly. The point of variables is so that another mod or an user can alter how a mod functions. They are a very powerful feature, and it can lead to a lot of neat possibilities for mods. Without variables, these settings would have to be read and written from a file, which would cause a lot of overhead and would make some features impossible.

All variables are numbers. No matter how they may appear in the configuration window, they are all internally represneted as numbers. Checkboxes simply toggle between 0 and 1 and dropdown lists choose a number corresponding to the location of the option on the list.

Variables can be used by a mod in one of two ways:

**Mini-Patch**
: When a variable is set to be a mini-patch, it automatically replaces whatever the given location is with the value of the variable. Code added by the mod can then test the number and branch to a certain subroutine or it can use the number directly in a calculation.

**Install Condition**
: During the installation of a mod, the variable is compared to another value. If the condition is true, the patch is processed normally. If the condition is false, the patch is skipped.

Variables are considered global. Any mod can use a variable set by another mod. For example, a level mod can see if the ”enable weather effects” variable of another mod is set, and if it not decide to not waste space installing code for rendering it's weather effects.

Variables can either be set or incremented by a certain amount. This action can only occur once per variable per mod, so don't worry about install conditions messing things up. As an example of how this would work, a mod for a custom level menu can define a certain variable for level mods to increment so that the menu knows how many entires to display.

## 2.5 Installation and Removal

Mods are installed sequentially. When a mod is installed, it first calculates the new values for any variables. After that, it goes through every patch in order and runs the operations described in 2.3.1. To assist in removing it later, it also takes some extra precautions. Specifically...

- When a part of a file is overwritten by a patch, the previous contents of that part of the file are recorded along with the UUID of the space that used that space.
- When an existing Clear space is converted to an Add space, the old space is renamed to (SpaceUUID).old.
- When a space from profile initlization is converted to a Clear space, the old space

Mods are also removed sequentially. When a mod other than the most recently installed is removed, every mod installed afterwards is removed and then readded. This simplifies the implementation of the mod loader compared to a random-access method.

The actions to take during removal are determined by taking the last space created and working backwards. When space is cleared durning installation, the removal process removes the clear space and restores the old add space if there was one. When space is added during installation, is recalls the old value of the bytes, writes them back in, then removes the add space and restores the old Clear space.

## 2.6 The SQL Database

The heart of the mod loader, so to speak, is an SQLite3 database containting every mod, patch, variable, file location and space. The choice of an SQL database might seem a bit puzzling at first, but it allows for the vast power the mod loader gives. The database is split up into several tables:

### 2.6.1 Mods

This table is a list of every mod installed. It is, essentially, what you see in the Details pane of the mod loader. It contains the following columns:

**UUID (TEXT)** A unique identifier for the mod. Does not have to be in standard UUID format.

**Name (TEXT)** The name of the mod. Used primarily in the user interface.

**Desc (TEXT)** Description of the mod.

**Auth (TEXT)** Creator of the mod.

**Ver (INTEGER)** Version of the mod.

**Date (TEXT)** Date mod was published in ISO-8601 format.

**Cat (TEXT)** Category of mod. See 3.1.7 for possible values.

**Pic (TEXT; Hex bytes)** Hexadecimal digits containing a Windows .BMP containing a preview picture for the mod loader.

### 2.6.2 Spaces

A list of spaces added or removed by mods or during profile initilization. See 2.2 for details.

**ID (TEXT)** A unique identifier to name the space so that mod developers can refer to it.

**Type (TEXT)** Either 'Add' or 'Clear', depending on the type of space.

**File (INTEGER)** The file that the space refers to. Use table 'Files' to convert to a path.

**Mod (TEXT)** The UUID of the mod that created the space.

**Start (INTEGER)** The first byte of the file described by the space.

**End (INTEGER)** The last byte of the file described by the space.

**Len (INTEGER)** Equal to End - Start.

**UsedBy (TEXT)** The mod that is currently using the space, if different than the creator.

### 2.6.3 Variables

Contains a list of public and private variables used by mods.

**UUID (TEXT)** A unique identifier for the variable. Does not have to be in standard UUID format.

**Mod (TEXT)** The UUID of the owning mod.

**Type (TEXT)** How the data stored in Value is represented. Possible values can be found at 3.3.4.

**PublicType (TEXT)** How the variable is displayed on the Config screen. If value is null, variable is considered "private" and not displayed on config screen.

**Desc (TEXT)** Human-readable description of the variable's purpose. Displayed in config screen.

**Value (INTEGER)** The number currently stored in the varaible.

#### 2.6.4 VarList

Small table used to store list entries for public varibles with PublicType set to "list".

**Var (TEXT)** UUID of the mod that uses this list.

**Number (INTEGER)** The number associated with the value in the list.

**Label (TEXT)** A description of what the entry in the list corresponds to in the mod's code or installation routine.

#### 2.6.5 Revert

Holds rollback data for mod uninstallation.

**PatchUUID (TEXT)** UUID of the patch (and consequently the space) that caused this entry to be created.

**OldBytes (TEXT; Hex bytes)** A hex dump of the previous contents of that part of the file.

#### 2.6.6 Files

Holds list of known files. Files are internally stored as numbers. They go up sequentially in the order that they are first referenced by a mod. 0 is always the main file and -1 is always the RAM.

**ID (INTEGER)** Number corresponding with the path.

**Path (TEXT)** Location of the file relative to the main file.

#### 2.6.7 Dependencies

Holds list of mod dependencies for uninstallation.

**ParentUUID (TEXT)** UUID of mod with dependency.

**ChildUUID (TEXT)** UUID of mod that is required by ParentUUID.

# 3 Creating Mods

This section describes how to compose the .JSON files that defines your mod. This will not help you produce content.

There are a couple things to note here:

- Numbers can either be in quotes or out of quotes. If they are out of quotes they must be in base 10. If they are in quotes, they can either be in base 10 (normal), base 8 (prefix with 0), or base 16 (prefix with 0x).
- All key names are case-sensitive
- The order of the elements does not matter, but the structure of objects and arrays does.

## 3.1 Defining Mod Metadata

Mod metadata is the bit of the mod that defines what it is. All the entries are places in the root object. An example of valid mod metadata is the following:

```
{
        "UUID": "cpucrashfix@invisibleup",
        "Name": "CPU Crash Fix",
        "Desc": "Fixes the crashing error that occurs
            when launching the game on modern systems.",
        "Auth": "InvisibleUp",
        "Ver": 1,
        "Date": "2015-05-06",
        "Cat": "Compatibility",
        "ML_Ver": "1.0.0",
        "Game": "SonicR1998"

        ...
}
```

### 3.1.1 UUID

A unique identifier for your mod. It does not matter what format it's in, but it is recommended that it's either a standard GUID (EX: AE28AAD8-2B2F-11E5-AD96-2484B6C90E9A) or in the format (modname)@(author). Either of those options should prevent duplicates from generic names.

### 3.1.2 Name

The name of your mod. Try to keep it short enough to fit in the mod loader's list but descriptive enough so that it won't be mixed up with other mods. For the above example, a bad name would be "CPU Fix" or "Crash Fix", as it's not certain what's being fixed with the CPU or what's causing the crash.

### 3.1.3 Desc

An informative description of what your mod does. It can be as long as you wish, but keep in mind that this is not a good place for a README file or documentation. The user needs to be able to read these and determine relatively quickly what the mod does without scrolling through an entire license agreement or list of system requirements.

### 3.1.4 Auth

The person or team responsible for creating the mod. You may use either your real name, or if you prefer you could use a username you use often on the internet. Whatever you decide to put down, make sure that an internet search for it can lead to information about you and your other mods. Do not, for example, put down "John Doe" as the author, as it will be impossible to trace to you in case an user needs support or is interested in other things you've made.

### 3.1.5 Ver

The version of your mod. It must be a whole number. It is suggested that you start at 1 and increment whenever you put out a new release. This allows users with an old version of your mod to upgrade using the mod loader without first uninstalling your mod.

### 3.1.6 Date

The date your mod was published in ISO 8601 format. (YYYY-MM-DD) This can be used by the user to determine if a mod needs updating or would be compatible with mods produced at a much later date.

### 3.1.7 Cat

This is the role your mod takes. While the mod loader does not really care what the value of this is, having this filled out allows users to easily sort mods by functionality. Valid values for this are:

**Essential** Mods who add features that would be considered mandatory in a more modern game. Examples include widescreen support, analog control, 60FPS support, etc.

**Compatibility** Mods whose sole purpose is to allow the game to run on more hardware configurations than it would without the mod.

**Code** Mods that add new subroutines for other mods to use or replace existing subroutines.

**Level** Mods that add new levels to the game.

**Character** Mods that add new playable characters to the game.

**Graphics** Mods that add or replace graphical elements for menus, existing levels, etc.

**Music** Mods that add or replace music.

**Sound** Mods that add or replace sound effects.

**Gameplay** Mods that change gameplay elements, altering the way the game is played.

**Total Conversion** Mods that completely replace a large portion of the game and introduce brand new grpahics, levels, characters, etc.

Some of these are not mutually exclusive. For example, a level mod is likely to include graphics for the character and custom code for the character's abilities. In this case choose the category that describes the primary reason a person would install your mod. (For example, a person is not going to install your character for it's textures. They will install it becuase they want to play as your character.)

#### 3.1.8 MLVer

The version of the mod loader your mod targets. This documentation describe version 1.0.0 of the Sonic R Mod Loader, so make sure that this value is equal to 1.0.0.

This value follows the pattern [MAJOR].[MINOR].[BUGFIX]. Increases in the major number represent compatibility-breaking enhancements or a major shift in the focus of the application's development. Increases in the minor number indicate the addition of new features, either in the interface or in this specification. Increases in the bugfix number add no new features; they just include stability or interface fixes.

## 3.2 Dependencies

If you are creating a mod in a well-established mod ecosystem, there is a good chance that you may wish to use code or resources from another mod. In this case you'll want to declare a dependency on that other mod. Declaring a dependency states that your mod will not function without the presence of the other mod. Do not use dependencies for optional features; use install conditions and the predefined variable "Active.(mod-uuid)" instead. Likewise, do not declare a dependency if all you need is free space or a bugfix to be installed. There may be a reason that particular mod is not installed.

Dependecies are declared in an array named "dependencies" in the root object, which contains objects which contain the keys and values.

```
"dependencies": [
        {         "Name": "Joe's Mod Framework",
                  "Ver": 14,
                  "Auth": "Team Joe & Co.",
                  "UUID": "{AE28AAD8-2B2F-11E5-AD96-2484
                     B6C90E9A}"          }
],
```

### 3.2.1 Name

The name displayed to the user when the dependency is missing. Make sure it matches the name of the actual dependency.

### 3.2.2 Ver

The minimum version of the dependency that your mod will work with. Do not set the minimum to the absolute newest version of the dependency, if it will run on a lower version. If functionality changes in earlier versions and you want to support them, use installation conditions and the predefined variable "Version.(mod-uuid)" to add code to support those versions.

### 3.2.3 Auth

The author name displayed to the user when the dependency is missing. Make sure it matches the author of the actual dependency.

### 3.2.4 UUID

The UUID of the dependency. This is what is looked for when checking if dependencies are met, so make sure this matches the dependency's UUID exactly.

## 3.3 Variables

Variables are arguably the most powerful feature of the Sonic R Mod Loader. Mastering their use is the key to creating great, powerful mods.

Variables are defined in an array named "variables" in the root object, which contains objects which contain the keys and values.

```
"variables": [
        {
        "UUID": "dresscolor.amyrose@invisibleup",
        "Type": "uInt32",
        "Value": 0,
        "Desc": "Color of Amy Rose's dress"
        "PublicType": "list",
        "PublicList": [
                {    "Number": "0x00FF0000",
                        "Label": Red      },
                {       "Number": "0x0000FF00",
                        "Label": Green   },
                {       "Number": "0x000000FF",
                        "Label": Blue     }
                ]
        }
],
```

### 3.3.1 UUID

This is a simple string containing a unique identifier for your mod.

Make certain your variables are named uniquely and descriptively. Variables are considered global. Because of this any mod can access a variable from any other mod. A good idea is to name your variables (variable-name).(mod-uuid), unless you need to make it generic to allow other mods to supplement or replace the functionality of yours.

### 3.3.2 Desc

A description of he variable. If the variable is public, this will be the text label that accompanies the control for the variable. Keep it short and use sentence case without an ending period.

### 3.3.3 Value

The default value for the variable. Must be a number within the range specified by the Type property. This is effectively pointless for public variables, but it's good practice, and it's also mandatory anyways.

It can also be a string in the format "+(num)" or "-(num)", where (num) is the amount to increment or decrement an existing variable by. If the variable named in the UUID property is not found, a new variable will be created. As such, if the value is in this format all properties listed after this one will only be parsed if a new variable is being created.

### 3.3.4 Type

Variables are always stored as numbers, but you can define how many bytes they use and in what format they're stored in. Possible values for this property include the following:

**Int8**
: This is a signed 8-bit integer. It has a range of -127 to 128. Use this for binary values or dropdown lists.

**uInt8**
: This is an unsigned 8-bit integer. It has a range of 0 to 255. Use this for small non-negative numerical values.

**Int16**
: This is a signed 16-bit integer. It has a range of -32768 to 32767. Use this for larger numerical values such as

**uInt16**
: This is an unsigned 16-bit integer. It has a range of 0 to 65535. Use this for larger numerical values such as

**Int32**
: This is a signed 32-bit integer. It has a range of -2147483648 to 2147483647. Use this for very, very large values. Note that at this size you may have trouble using the input in your ASM code. Make sure the operation you are using supports integers of this size.

**uInt32**
: This is an unsigned 32-bit integer. It has a range of 0 to 4294967295. Use this for pointers or very, very large values. Note that at this size you may have trouble using the input in your ASM code. Make sure the operation you are using supports integers of this size.

**IEEE32**
: This is an IEEE-754 32-bit float value. It supports a decimal with a significant figure of roughly 6 to 9 places. This cannot be used directly by most x86 operations, including cmp and the arithmetic functions. You must load this into an x87 FPU register in order to do anything meaningful with it.

**IEEE64**
: This is an IEEE-754 64-bit double-precision float value. It supports a decimal with a significant figure of roughly 15 to 17 places. This cannot be used directly by most x86 operations, including cmp and the arithmetic functions. You must load this into an x87 FPU register in order to do anything meaningful with it.

### 3.3.5 PublicType

Optional value describing how the variable is displayed on the mod's Config screen. If ommited, variable will not be displayed on the Config screen. Possible values for this property are the following:

**numeric** Represents a number. Displayed using the up-down numerical control is possible.

**list** Like an C enum; represents a list of labels that corresponds with numbers. Displayed using a ListBox control if possible.

**checkbox** Represents a value that can be toggle on (1) or off (0). Displayed using a checkbox control if possible.

Try to opt for numeric or checkbox if possible. The example above would be much better if the red, green, and blue componets of the color were seperate variables with numeric controls.

### 3.3.6 PublicList Array

Optional array listing the values for the listbox displayed on the mod's configuration screen. Ignored unless PublicType is "list".

**Number** The number associated with the value in the list.

**Label** A description of what the entry in the list corresponds to in the mod's code or installation routine.

### 3.3.7 Predefined Variables

For your convienence, a few variables are automatically defined for you.

**Version.ModLoader** uInt32. Set to the mod loader's version number in the format [0x00MMmmbb], where M = Major Version, m = Minor Version and b = Bugfix version.

**Active.(mod-uuid)** uInt8. Set to 1 if (mod-uuid) is installed. Use to determine the presence of a mod.

**Version.(mod-uuid)** uInt16. Set to the version number of the the mod (mod-uuid). Use to branch installation or code on different versions of a mod with different features.

**Exists.(path)** uInt8. Set to 1 if a file exists on the system. Use to determine the presence of a file.

If a variable is undefined, it's value will be reported as 0.

## 3.4 Patches

Patches are the actual operations that your mod applies on the game's files.

Patches are defined in an array named "patches" in the root object, which contains objects which contain the keys and values. This may look like a lot, but most the arguments are optional depending on the contents of Mode.

```
"patches": [
        {
                "Mode": "Repl",
                "File": "SONICR.EXE",
                "Start": 460164,
                "End": 460170,
                "DestFile": null,
                "DestStart": null,
                "DestEnd": null,
                "AddType": "Bytes",
                "Value": "B8310000009090",
                "Comment": "NOPs out timer code"
                "ConditionVar": "timerdisable.
                    cpucrashfix@invisibleup"
                "ConditionOp": "NOT EQUALS"
                "ConditionValue": 0
        }
],
```

### 3.4.1 Mode

What the patch does. The value of this is referred to as the "operation". Can be one of 6 values, ADD, CLEAR, RESERVE, REPL, COPY or MOVE. For information on what the different operations do, see 2.3.

### 3.4.2 File

The path to a file within the game's directory, relative to the main EXE's location. It can also be the special path ":memory:" to access a virtual file of infinite length dedicated to storing RAM.

For MOVE and COPY this is the source file.

### 3.4.3 Start

The lowest possible byte that is okay to modify.

For every operation except ADD, this is the lowest byte that will be modified. For ADD, this is the lowest byte to look for a clear space.

If this value is omitted the operation is considered to be whole-file. See 2.3.4 for more information.

This value can also be the UUID of an existing space. Using the UUID of an existing space will set this value to the beginning of the space, and set the End value to the end of the space.

### 3.4.4 End

The highest possible byte that is okay to modify.

For every operation except ADD, this is the highest byte that will be modified. For ADD, this is the highest byte to look for a clear space.

Omit this if you are using the UUID of an existing space for Start.

#### 3.4.5 DestFile

The file to transfer values from File to during a COPY or MOVE operation. Relative to the folder containing the main file.

#### 3.4.6 DestStart

The location of the lowest possible byte to COPY or MOVE to in DestFile.

This value can also be the UUID of an existing space. Using the UUID of an existing space will set this value to the beginning of the space, and set the DestEnd value to the end of the space.

Omit this if using any operation other than COPY or MOVE.

#### 3.4.7 DestEnd

The location of the highest possible byte to MOVE or COPY to.

Omit this if you are using the UUID of an existing space for DestStart. Omit this if using any operation other than COPY or MOVE.

#### 3.4.8 AddType

For ADD and REPL operations, indicates the format the contents of Value. Possible values are the following:

**Bytes**
Raw bytes in hexadecimal form to be inserted into the given space.

**Path**
A path to a file inside the mod's folder or archive to be inserted in whole into the given space.

**UUIDPointer**
The UUID of an add space, optionally followed by a (+ num) or a (-num). The data to be added is a 32-bit pointer corresponding with the space's (Start) value, which is than added or subtracted by the given value. This allows you to call subroutines or access memory variables located in other patches, and perhaps from other mods entirely.

**VarValue**
The UUID of a variable. The data to be added is the contents of the variable, in the datatype specified when the variable was first created. Use for calculations or code branching that is dependent on the status of other installed mods.

Omit this for all other operations.

### 3.4.9 Value

The content to be added, in the format indicated by the AddType value. See the AddType listing for what to insert here.

### 3.4.10 Comment

A comment describing what the patch does. Never displayed; for mod creator's reference only.

### 3.4.11 ConditionVar

The UUID of the variable to be compared to ConditionConst by the criteria in ConditionOp.

Required if ConditionOp is present. Do not include otherwise.

### 3.4.12 ConditionConst

The value to be compared to ConditionVar by the criteria in ConditionOp.

Required if ConditionOp is present. Do not include otherwise.

### 3.4.13 ConditionOp

The condition to test to determine if patch is to be parsed.

Can be one of:

**EQUALS**
Parse patch if ConditionVar is equal to ConditionConst.

**NOT EQAULS**
Parse patch if ConditionVar is different from ConditionConst.

**LESS**
Parse patch if ConditionVar is less than ConditionConst.

**LESS EQUAL**
Parse patch if ConditionVar is less than or equal to ConditionConst.

**GREATER**
Parse patch if ConditionVar is greater than ConditionConst.

**GREATER EQUAL**
Parse patch if ConditionVar is greater than or equal to ConditionConst.

This value is optional.

# 4 Adding Support for Additional Games

Despite being called the Sonic R Mod Loader, this program can load mods for games that aren't Sonic R. (It was a rather shortsighted name, in hindsight.) The program is explicitly designed so that defining custom games or applications will be easy. Game definitions can be shared as easily as mods and only one version of the mod loader will need to be maintained for every possible game.

## 4.1 Game Metadata

The information required to load a game is stored in a .JSON fie that shares a lot of similarities with a mod.

```
{
"GameName": "Sonic R (PC, 1998)"
"GameUUID": "SonicR_PC1998",
"GameEXE": "SONICR.EXE",
...
}
```

### 4.1.1 GameName

A human-readable name for the game. If you can, keep it in the following format:

```
GameName Version [Platform] (Publisher, Year, Country)
EX: Shinobi REV1 [Amiga] (Sega, 1989, JPN)
```

Do not include information that is not required. If there was only one worldwide release on one year on one platform by one publisher with no revisions, you should only put the title.

### 4.1.2 GameUUID

A unique identifier for the game. This will need to be entered any time anyone makes a mod for the game, so keep it typeable. Also do not put the author name after an @ like you would with a mod. The best advice is to copy the "GameName" value and strip all punctuation, replacing [ and ( with underscores.

### 4.1.3 GameEXE

This value, referred throughout the documentation as the "main file", is the file that will be executed when the "run" button is pressed in the loader interface. It is also the file that will be checked to determine the game's version and validity. Even if another file stores most of the game's data, make sure the program that is launched is set as the main file.

## 4.2 Whitelist

The whitelist is a list of the filesize and checksums of all valid main files. For slightly altered versions (common anti-piracy hacks, commonly-applied bugfixes, etc.) you can define several different valid files, along with the UUID of a patch that would enable that functionality on an otherwise "vanilla", or unmodded, file.

If there is a major change (file size gets cut in half, all functions get relocated, or even if just a couple of functions get relocated) create a new game definition instead.

```
...
"Whitelist": [
        ...
        {
                "Desc": "CPU crash bugfix",
                "Size": 1263104,
                "ChkSum": "0xe251e7fc",
                "Mods": ["cpucrashfix@invisibleup"]
        }
],
        ...
```

### 4.2.1 Desc

A human-readable description of the variant.

Keep it short and sweet. If it's the vanilla EXE, call it "Vanilla EXE". If it's an antipiracy hack, call it "Bob's AntiPiracy Hack". Don't overthink it.

### 4.2.2 Size

The size of the file in bytes. In theory these should be the same for all variants, but some games can append or subtract garbage from the end. If the different file causes the functions to be in different locations, create a new game definition file.

#### 4.2.3 ChkSum

The checksum of the file. It's supposed to be CRC32, but it may not be compatible with any checksum utilities you have. For 100% accuracy, use the "CHKSUM" file in the archive the program was distributed in.

#### 4.2.4 Mods

A list of "installed" mods. These will not be granted entries on the mod loader screen, but their installation will be blocked. Technically you could keep this blank and not have any consequences, but it's nice insurance.

## 4.3 KnownSpaces

Here's the useful part. KnownSpaces is a list of every known subroutine, resource or memory block that a mod developer can access. You provide this. These entries are directly translated into Add spaces that mods can use later. (Effectively these are patches that are always parsed with the RESERVE operation.)

Typically this information is derived from a disassembly. If you're using IDA to disassemble, a BASH script is provided that runs the .MAP output from IDA and spits out a .CSV containing every subroutine. (It can be edited to spit out RAM instead, if you need that.) Use a CSV-¿JSON converter like http://www.convertcsv.com/csv-to-json.htm to convert it to the proper format from there.

```
...
"KnownSpaces": [
        {
                "Start":76948,
                "End":77212,
                "File":"SONICR.EXE",
                "UUID":"Game_RenderFog"
        },
        ...
],
...
```

### 4.3.1 Start

The first byte of the space. Must be a number less than the length of the file.

### 4.3.2 End

The last byte of the space. Must be a number less than the length of the file.

### 4.3.3 File

The path to the file this space applies to, realtive to the main file. It can also be the virtual file ":memory:", an infintely large file that represents the non-allocated memory.

### 4.3.4 UUID

A unique identifier for the space. If you're deriving this list from a disassembly, just use the function name.