Help for Sonic R Mod Loader 1.0.0

InvisibleUp

February 15, 2017

This is the documentation for the Sonic R Mod Loader. It covers 4 main areas:

Using the Mod Loader
This section deals with how to operate the mod loader, installing and uninstalling mods.

How the Mod Loader Works
This section describes the mechanics of the basic functionality of the mod loader. It explains how mods are stored and installed.

Creating Mods
This section describes how to create a mod and the files that comprise a mod.

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 extremely modest system requirements. So modest, it will run on anything Sonic R will run on, and then some. The program has been confirmed to work out of the box with the following system:

Operating System Windows 98/NT 4.0 or greater

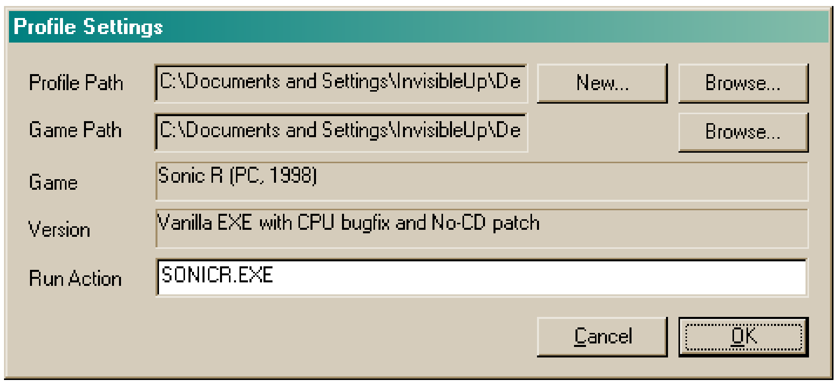
CPU Pentium (i586) processor or greater

RAM 16MB or greater

Disk 2MB free space plus at least 10MB for mods.

Display SVGA display (800x600x16)

1.2 Creating a Profile


Profile Settings

Profile Path	C:\Documents and Settings\InvisibleUp\De
Game Path	C:\Documents and Settings\InvisibleUp\De
Game	Sonic R (PC, 1998)
Version	Vanilla EXE with CPU bugfix and No-CD patch
Run Action	SONICR.EXE

Figure 1: The profile editor dialog

Profiles are a simple way of keeping different game installations from mingling and messing your mods up. (Note, however, that there can only be one set of mods per game directory.) You will need to create a profile when you start the program for the first time.

Click the “New” button next to “Profile” to add a new profile. The program will prompt you for a place to save your profile. When you want to load it up later (using the “Browse” button), select this file.

After that, you will need to set the installation path of your game. Use the “Browse” button next to “Game” to select the folder where your game is installed. The program will automatically detect which game and version is installed.

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.

The Run Path is the program launched when you press the “Run” button in the main window. By default, the “Run Path” is set to the game’s executable. If you need to change this for, ex: compatibility reasons, edit this text box.

Once you are done, press “OK” to save your profile to the file listed in the “Profile” textbox.

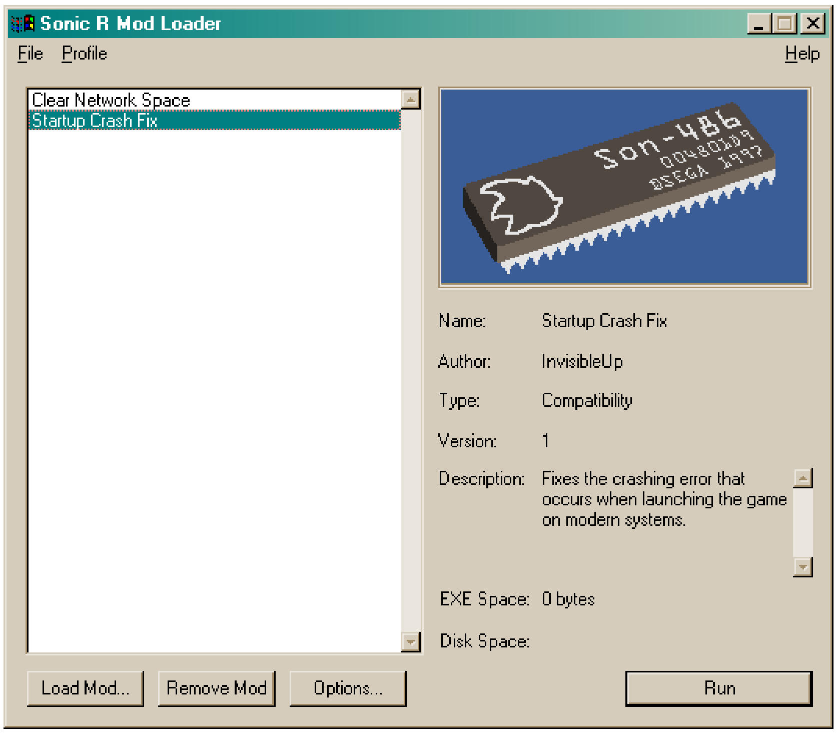
1.3 The Mod Loader Interface

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.

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 whichever mod is selected.

2. **Preview**
   This is an optional picture provided by the mod’s developer meant to quickly identify 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 a distinctive picture is used instead.

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: Startup Crash 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:

Figure 2: The primary interface of the Sonic R Mod Loader

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. If you're having a trouble with a mod, ask the mod's author, NOT the developer of the mod loader.

Category
: This is the role that the mod takes on. A list of category types can be found at 3.3.7.

Version
: The version number of the mod. Some mods may require that a mod of a certain version is installed.

Info
: A brief outline of the mod's contents and purpose.

There are also 4 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 1.6 for more information.

Options...
: Shows a list of all public variables that the user can customize. See 1.5 for details.

Run
: Runs the file that “Run Path” in the profile editor refers to. Make sure that, if you need a program like DXWnd to make the game work properly, you run it 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, as the mod loader does not support directly downloading mods from 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 malware into a mod. If in doubt, ask others on the site you got it from before installing 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. This will be a ZIP 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 uninstalled. If this occurs, you will need to either uninstall some mods or 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 game 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 are 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.

All public variables are always enabled and displayed independently, but keep in mind a mod developer could make it so that some are selectively ignored depending on the value of another.

Keep in mind that some public variables 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 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 alert you. If this happens, you will need to remove those mods.

If 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 a highly simplified model of the inner workings of the Sonic R Mod Loader. This will be useful reading if you're planning on making 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. This .ZIP archive is extracted by the user before installation, for the sake of simplifying the program. Support for installing from .ZIP files may be added in the future.

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

'Split' spaces record the "head" and "tail" spaces after a space has been split by an internal mod loader process. They are created when an 'Add' space is spliced into into a larger 'Clear' space. This causes only the portion of the 'Clear' space required by the new 'Add' space to be used. The remainder is seperated into new 'Clear' spaces. 'Split' spaces exist so that the mod loader can "rewind" and merge the spaces back together during mod uninstallation.

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 in 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 a a unique identifier, an operation type and (depending on the type) a range of source bytes, the source file, a range of destination bytes and file, and a data value for insertion.

2.3.1 Operations

There are 6 different operations a patch can preform:

ADD
: Splices an 'Add' space containing the given data into an existing 'Clear' space.

CLEAR
: Moves the given range of bytes to a new 'Clear' space.

RESERVE
: Splices an 'Add' space spanning the given range of bytes into an existing 'Clear' space.

REPL
: Replaces the contents of the given range of bytes with a different value.

COPY
: Copies the contents of the given range of bytes to a new location.

MOVE
: Moves the contents of the given range of bytes to a new location, leaving a 'Clear' space behind.

These 6 operations should accomplish just about anything you would need to do. Any advanced operations should be possible with a sequence of these operations. (If not, feel free to send a feature request to the project's GitHub page.)

⚠ Note that the mod loader does not yet support adding 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.

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, or mods with assets that must be loaded. This is done by specifying a source and destination file for each operation.

2.3.2 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. This is done by omitting the start and end byte ranges.

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. Variables exist 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 likely have to be read and written from a file, which would cause a lot of overhead and would make some features impossible.

In addition, all spaces are given a variable that contains their starting address. Custom code can use this to call functions or use data from other mods or the base game.

All variables are numbers. No matter how they may appear in the configuration window, they are all internally represented as numbers. Check boxes simply toggle between 0 and 1 and drop down 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 are set or altered before any patches are installed. 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 entries 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 unique identifier and version of the space that used that space.
- When a space is replaced, it is marked as "used" and a new space is created with a higher version number. Only the highest version number is considered during installation.
- If a space only takes up part of a CLEAR space, a SPLIT psuedo-space is created that documents what space got split up.

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

During removal, the loader runs through the spaces in reverse. For pretty much every space, this just means removing the higher versioned space and writing the stored contents back to the file.

3 Creating Mods

This section describes how to compose the .JSON files that defines your mod. Note that you'll still need to do the hard part of creating content for the mod. (This manual can not and will not teach you how to do that.)

There are a couple very important 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, base 8 (prefix with 0), or base 16 (prefix with 0x).
- The order of the elements does not matter, but the structure of objects and arrays does.

3.1 Files

The structure of your ZIP file must be as listed:

info.json All patch and metadata information. The only required file. The rest of this section will deal solely with this file.

preview.bmp Optional preview to be displayed in the interface. MUST be in Windows BMP format, although bit depth does not matter. Recommended size is 235x135. Any excess space in the preview window will be filled with the color of the top-leftmost pixel.

(any other files) You can include other files for use in your patches. The mod loader does not care how these are stored.

3.2 Expressions

Many numerical fields within the mod metadata are actually parsed expressions. This allows for some neat stuff, like setting variables based on other variables (including itself!) or several other things I couldn't possibly think of.

Here are some examples:

```
35
36 + 78
(1 + 2) * (3 + 4) * 6 + 7
\$ var.mod@user
(len @ list.bin) / 4
\$ var1 == 0
0x4567 ^ 0x1234
```

3.2.1 Numbers and References

You can have just plain old numbers, in base 10, 16, or 8. Nothing special here.

You can also refer to either the value of a variable or certain properties of a file. To do this, enter a dereferencing operator (listed below) followed by the name of a file or variable.

3.2.2 Operators

Operators for expressions follow standard infix notation, separated by EXACTLY ONE space. Below is a table of supported operators.

Op	Description	Notes	Order
+	Addition		3
-	Subtraction		3
*	Multiplication		2
/	Division		2
%	Modulo	Integers only	2
|	Bitwise OR	Integers only	7
&	Bitwise AND	Integers only	7
ˆ	Bitwise XOR	Integers only	7
˜	Bitwise NOT	Integers only	1
<<	Right shift	Integers only	4
>>	Left shift	Integers only	4
==	Equal	Gives 0 if true, 1 if false	6
!=	Not equal	Gives 0 if true, 1 if false	6
<=	Less than or equal	Gives 0 if true, 1 if false	5
>=	Greater than or equal	Gives 0 if true, 1 if false	5
<	Greater than	Gives 0 if true, 1 if false	5
>	Less than	Gives 0 if true, 1 if false	5
&&	Logical AND	0 if both sides 0, 1 if not	8
||	Logical OR	0 if any side 0, 1 if not	8
$	Value of variable	Unary. Type will be ignored	1
@	Deref file from game dir	Unary. Must follow 'len' or 'crc32'	1
#	Deref file from mod dir	Unary. Must follow 'len' or 'crc32'	1
crc32	CRC32 checksum of file	Unary. Must precede @ or #	1
len	Length of file in bytes	Unary. Must precede @ or #	1
exists	0 if variable or file exists	Unary. Must precede $, @ or #	1
(	Left grouping parenthesis		0
)	Right grouping parenthesis		0

Operators are grouped in order of decreasing precedence. Ex: (4 + 5 * 4) = 4 + (5 * 4). All operators excluding '~' are left-associative. The operators '~', 'crc32', 'len', 'exists', '$', '@', and '#' are all unary instead of binary. No short-cicruting for logical or is implemented. (Generally speaking, if you follow the rules of the C programming language, you'll be fine.)

File dereferences will return an error unless they're immediately preceded by the 'len' or 'crc32' operators. This may change in future versions. Also 'exists' must be followed by a dereferenced file or variable name.

Some operators only work on integer types. You'll be fine in most circumstances, as expressions are only evaluated as (double-precision) floating point when they're the value of a floating point variable. Otherwise values are internally parsed as an Int32 or uInt32, depending on whatever is needed. All variations will be noted in the field descriptions.

3.3 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": "CpuStartupFix@invisibleup",
    "Name": "CPU Startup Crash Fix",
    "Info": "Fixes the crashing error that occurs when
       launching the game on modern systems.",
    "Author": "InvisibleUp",
    "Version": 1,
    "Date": "2016-05-06",
    "Category": "Compatibility",
    "ML_Ver": "1.0.0",
    "Game": "SonicR_PC_1998"
    ...
}
```

3.3.1 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.

3.3.2 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.

3.3.3 Info

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.

3.3.4 Author

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. (And definitely do not steal another person's name. Like "InvisibleUp". That's my name. I'd be very upset if you stole my name.)

3.3.5 Version

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. If you produce a release with functions that are binary-incompatible (different arguments for calling, etc.) with previous versions, you should change the UUID to prevent program crashes and other nastiness.

3.3.6 Date

The date your mod was created 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. (As of version 1.0.0, this isn't actually listed in the interface. This is likely to change in future versions.)

3.3.7 Category

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

Essential Mods that add features that would be considered mandatory in a more modern game. Examples include widescreen support, shaders, higher framerate, etc.

Compatibility Mods whose sole purpose is to allow the game to run on more hardware configurations.

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 graphics, levels, characters, etc.

Some of these are not mutually exclusive. For example, a character 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. (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.)

3.3.8 ML_Ver

The lowest version of the mod loader your mod targets. This documentation describes 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 user interface or in this specification. Increases in the bugfix number add no new features; they just include stability or interface fixes.

If you plan on using features from later versions while still being compatible with an older version, use install conditions against the predefined variable "Version.MODLOADER@invisibleup".

3.4 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",
          ''Version": 14,
          ''Author": ''Team Joe & Co.",
          ''UUID": ''{AE28AAD8-2B2F-11E5-AD96-2484
            B6C90E9A}"
    }
],
```

3.4.1 Name

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

3.4.2 Version

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.

3.4.3 Author

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

3.4.4 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.

3.5 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",
    "Default": "0x00FF0000",
    "Update": "0x0000FF00",
    "Info": "Color of Amy Rose's dress"
    "PublicType": "list",
    "PublicList": [
        { "Value": "0x00FF0000",
        "Label": Red },
        { "Value": "0x0000FF00",
        "Label": Green },
        { "Value": "0x000000FF",
        "Label": Blue }
    ] }
],
```

(This is a rather contrived example, but it gets the point across)

3.5.1 UUID

This is a simple string containing a unique identifier for your mod. If a variable already exists with this UUID, it will be skipped over. (Do not rely on the behavior; the variable may suddenly change values when mods are reinstalled.)

Make certain your variables are named uniquely and descriptively. Variables are considered global; any mod can access a variable from any other mod. A good idea is to name your variables (variable-name).(mod-uuid) to avoid conflict.

3.5.2 Info

A description of the 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.

3.5.3 Default

[EXPRESSION, type dependent on Type field]

If the variable is not yet defined, the Default expression is parse and set as the variable's new value.

3.5.4 Update

[EXPRESSION, type dependent on Type field]

If the value already exists, it's value will be set to the value of this expression. If this field is omitted, the variable will keep it's current value.

3.5.5 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.

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

Int32
: This is a signed 32-bit integer. It has a range of -2147483648 to 2147483647. Use this for very large values. Note that at this size you may need to load the value into a register instead of using it as an immediate operand.

uInt32
: This is an unsigned 32-bit integer. It has a range of 0 to 4294967295. Use this for pointers or very large values. Note that at this size you may need to load the value into a register instead of using it as an immediate operand.

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.

3.5.6 PublicType

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

Numeric Represents a number. Displayed using the up-down numerical control. (Windows: IEEE and Int32 values are displayed as textboxes)

List Like an C enum; represents a list of labels that corresponds with numbers. Displayed using a drop-down selector.

Checkbox Represents a value that can be toggle on (0) or off (1). Displayed using a checkbox.

3.5.7 PublicList Array

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

Value [EXPRESSION, type dependent on Type field] 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.

3.5.8 Predefined Variables

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

Version.MODLOADER@invisibleup 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 0 if (mod-uuid) is installed. Use to determine the presence of a mod.

Version.(mod-uuid) uInt32. Set to the version number of (mod-uuid). Use to branch installation or code on different versions of a mod with different features.

Start.(patch-uuid).(mod-uuid) uInt32. Equal to the patch's (Start) value. If the add space refers to a Win32 EXE or DLL, the value is a location in memory when the executable is loaded. Otherwise it is a simple file offset.

End.(patch-uuid).(mod-uuid) uInt32. Equal to the patch's (End) value.

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

⚠ Be careful with undefined Checkbox variables. This will default to On, so make sure that whatever 'On' corresponds to is non-destructive.

3.6 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": [
    {
        "ID": "TimerNOP.cpufix@invisibleup",
        "Comment": "NOPs out timer code",

        "Mode": "REPL",
        "File": "SONICR.EXE",
        "FileType": "PE",
        "Start": 460164,
        "End": 460170,
        "Len": 6,

        "SrcFile": "data.bin",
        "SrcFileType": null,
        "SrcFileLoc": "Mod",
        "SrcStart": "0x0000",
        "SrcEnd": "0x0100",

        "AddType": "Bytes",
        "Value": "B8310000009090",

        "Condition": "\$ timerdisable.
           cpucrashfix@invisibleup != 0",

        "MiniPatches": [
            {
                "Var": "example.mod@invisibleup",
                "MaxLen": "2"
                "Pos": "0x03"
            },
            ...
        ]
    }
],
...
```

Note that the above patch contains many fields that will never be looked at or are otherwise nonsensical. It's merely an example of syntax, not how to write a good patch.

3.6.1 ID

This is a unique identifier for your patch. It is optional, as if you don't provide an ID one will be automatically generated for you for internal bookeeping purposes. You should provide one if you wish for other mods (or other patches within your mod) to call or reference the content added by the current patch.

Omit this for REPL operations.

3.6.2 Mode

What the patch does. The value of this is referred to as the "operation". Can be one of the six following values:

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 ID of (ID), or an autogenerated one if (ID) 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). This marks the space as ready to be used by any mod, and can no longer be referenced by ID.

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 (SrcStart) to (SrcEnd) as (Value) and applies an ADD operation from (Start) to (End). Fails on the same conditions that ADD fails.

MOVE

Applies an (ADD) operation from (Start) to (End) with the contents of (SrcStart) to (SrcEnd), then applies a CLEAR operation from (SrcStart) to (SrcEnd). Fails on the same conditions that CLEAR and ADD fails.

When the patch specifies (File) and/or (SrcFile), but not (Start), (End), (SrcStart) or (SrcEnd), (Start) is assumed to be 0, (End) the end of (File), and similar for (Src*).

3.6.3 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 destination file.

3.6.4 FileType

The type of file that (File) points to. The value of this will transform (Start) and (End) for internal use.

PE

Compute (Start) and (End) as memory locations when a Windows executable is loaded. Relocation tables are not currently supported.

(anything else)

Raw file offset as viewed in a hex editor.

3.6.5 Start

[EXPRESSION, uInt32]

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 and (End) are omitted, the operation is considered to be whole-file. See 2.3.2 for more information.

If only (End) is ommited, (Start) is assumed to refer to an existing space. (End) and (ID) are therefore set to whatever they're defined as for the existing space. (ID is only set if it's not already defined.) This is a convenient way to apply an operation (such as CLEAR or REPL) to the entirety of an existing space.

3.6.6 End

[EXPRESSION, uInt32] 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.

3.6.7 Len

[EXPRESSION, uInt32]
The length to allocate for an ADD or RESERVE operation. If (Len) is not long enough to fit all the bytes in (Value), the length required by (Value) will be used instead.

3.6.8 SrcFile

The file to transfer values from during a COPY or MOVE operation.

3.6.9 SrcFileType

The type of file that (SrcFile) points to. See entry for (FileType) for details.

3.6.10 SrcFileLoc

Determines where (SrcFile) is located.

Mod
Pull (SrcFile) from the mod's installation .ZIP.

(anything else)
(SrcFile) is in the game's installation directory.

3.6.11 SrcStart

[EXPRESSION, uInt32]
The location of the lowest possible byte to COPY or MOVE from in SrcFile (or File, is SrcFile is missing). Omit this if using any operation other than COPY or MOVE.

If (SrcEnd) is ommited, (SrcStart) is assumed to refer to an existing space. (SrcEnd) is therefore set to whatever they're defined as for the existing space.

3.6.12 SrcEnd

[EXPRESSION, uInt32]

The location of the highest possible byte to MOVE or COPY from in SrcFile (or File, is SrcFile is missing). Omit this if using any operation other than COPY or MOVE.

3.6.13 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.

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.

Expression
Parse an expression as a uInt32. If the value is greater than the allotted length allows for, the most significant bits will be dropped. Exercise caution when using with floating point variables.

Omit this for all other operations. To add the contents of another file, use the COPY operator and set SrcFile to the file to add from.

3.6.14 Value

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

3.6.15 Comment

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

3.6.16 Condition

[EXPRESSION, uInt32]

Installs the variable if and only if the condition evaluates to 0. This value is optional. To do an 'else' statement, create another patch with the opposite condition set.

⚠ If a variable is undefined, it will be equal to 0 and therefore evaluate as True. Same if the variable is defined, but is equal to 0. Use the idiom "exists $ varname && $ varname" to return true if and only if the variable is defined as 0.

3.6.17 MiniPatches

Array of places to replace a chosen section of a path.

Each item of the array is an object. Within each object are the following values:

Var

ID of the Variable to splice in

Expression

Expression to parse and splice in. If both 'Var' and 'Expression' are defined, 'Expression' will be used.

MaxLen

Maximum length of the data, in bytes. If it is over this limit, the most significant bits will be dropped.

Pos

Offset from the patch where the data is to be spliced in.

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 short-sighted name, but I haven't come up with anything better in the past year...) The program was explicitly designed so that defining custom games or applications would be easy. Game definitions can be shared as easily as mods. Also, only one version of the mod loader will need to be maintained for every game possible.

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_PC_1998",
"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. Our recommendation is to copy the "GameName" value and replace all punctuation with underscores. (EX: Shinobi_REV1_Amiga_Sega_1989_JPN)

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 Startup Crash fix",
        "Size": 1263104,
        "ChkSum": "0xe251e7fc",
        "Mods": ["cpufix@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 anti-piracy hack by Bob, call it "Bob's Anti-Piracy 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 data from the end. If the different file causes the functions to be in different locations, create a seperate game definition file for it.

4.2.3 ChkSum

The checksum of the file, in CRC32 format. You can calculate this with many different programs, including 7-Zip on Windows and 'crc32' on Mac/Linux.

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 takes an .IDC dump as an input and compiles it (because surprisingly enough an .IDC file is valid C) against a dummy library that produces a JSON object to copy/paste into your game configuration.

```
...
''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 making this list from a disassembly, just use the function name.