riffing
remix and dub
mistakes
with a very special intro by

RLdev: a visual novel development kit

Dr. Brian Oblivion

Version 1.45 — 2021-06-23

At some point in 2007, although some say 2008, which could even be as late as 2009, but everyone agrees that it had to be way before 2010....

Well, probably, was it?

[Start again]

At some point which could be debated once nothing remained to be done anywhere in the known universe, a naive and handsome knight arrived on the scene finding sheafs of code lying out in the open air at the edge of a great lake. "Hi there, all alone little fella?" R23 said to the code. Looking around, not a soul in was sight as far as the eye cared to see. "Yoink!" ended the conversation and as before the lake resumed lapping at the shore trying in vain to reach the empty clearing at its edge. What happened after that and what relevance it has here is unknown.

RLdev program and documentation copyright © 2006 Hæleth.

This is a development release. This software is incomplete; it is known to contain some bugs that may cause it to generate unsound code, and many implementation details are subject to change. You should not use this software for any purpose whatsoever unless you are sure you know what you're doing.

Contents

A Very Special Introduction iii

Acknowledgements iv

Preface v

1 Addendum 2011 1

2 Using RLdev 2
- 2.1 Kprl: the archiver/disassembler . . . 2
- 2.2 Rlc: the compiler . . . 5
- 2.3 Vaconv: the graphic converter . . . 8
  - 2.3.1 Bitmap formats . . . 9
  - 2.3.2 Advanced G00 features . . . 9
- 2.4 RlXml: the auxiliary data converter . . . 10

3 Getting started 11
  - 3.0.1 The project header file . . . 11
  - 3.0.2 Memory allocation . . . 11

4 Kepago 13
- 4.1 Lexical structure . . . 13
- 4.2 Structure of a file . . . 13
- 4.3 Expressions . . . 14
- 4.4 Statements . . . 14
  - 4.4.1 Blocks and scopes . . . 14
  - 4.4.2 Labels . . . 15
  - 4.4.3 Variable declarations . . . 15
  - 4.4.4 Constant declarations . . . 17
  - 4.4.5 Inlines/Macros . . . 18
  - 4.4.6 Directives . . . 18
  - 4.4.7 Conditional compilation . . . 20
  - 4.4.8 Control structures . . . 21
  - 4.4.9 Function calls . . . 23
- 4.5 String handling . . . 23
  - 4.5.1 Displaying text . . . 23
  - 4.5.2 Usable characters . . . 24
  - 4.5.3 Control codes . . . 25
  - 4.5.4 Names . . . 27

4.5.5 Glosses . . . 28
4.5.6 Lineation . . . 28
4.6 Resource files and resource strings . . . 28
4.6.1 Resource file syntax . . . 29
4.6.2 Additional control codes . . . 29
4.6.3 Anonymous references . . . 30
4.6.4 Using resource strings . . . 31

5 The RealLive system **32**
5.1 Overview . . . 32
5.1.1 Targets and versions . . . 32
5.2 Scenarios . . . 33
5.3 Debugging . . . 34
5.4 Memory . . . 34
5.4.1 Integers . . . 34
5.4.2 Strings . . . 35
5.4.3 Call variables . . . 36
5.5 The system command menu . . . 36
5.6 Extension DLLs . . . 40
5.6.1 Using an extension DLL . . . 40
5.6.2 The extension DLL interface . . . 40

6 The Kepago/RealLive API **44**
6.1 Introduction . . . 44
6.2 Duplicates . . . 45
6.3 Compatibility . . . 45
6.4 Compiler functions . . . 45
6.4.1 Initialisation . . . 45
6.4.2 Symbolic manipulation . . . 46
6.4.3 Compile-time `gameexe.ini` access . . . 49
6.5 Flow control . . . 49
6.5.1 Termination . . . 49
6.5.2 Inter-scenario jumps . . . 49
6.5.3 Local jumps . . . 50
6.5.4 Call stack and execution state . . . 52
6.5.5 Interrupts . . . 52
6.6 Variable manipulation . . . 53
6.6.1 Integers . . . 53
6.6.2 Arrays . . . 55
6.6.3 Integer blocks . . . 55
6.6.4 Strings . . . 56
6.6.5 Name variables . . . 59
6.7 Input . . . 59
6.7.1 Selections . . . 59
6.7.2 Text boxes . . . 61
6.7.3 Mouse input . . . 61
6.7.4 Keyboard input . . . 62
6.8 Text window controls . . . 63
6.8.1 Pausing and breaking . . . 63
6.8.2 Moving text . . . 63

6.8.3 Appearance . . . 64
6.8.4 Text speed . . . 66
6.8.5 Clearing text windows . . . 66
6.8.6 Window animations . . . 67
6.8.7 Character portraits . . . 67
6.9 Sound . . . 68
6.9.1 Sound settings . . . 68
6.9.2 Music . . . 68
6.9.3 Sound effects . . . 70
6.9.4 Interface sounds . . . 71
6.9.5 Voices . . . 72
6.10 Graphics . . . 74
6.10.1 Screen settings . . . 74
6.10.2 Device contexts . . . 75
6.10.3 Default bitmaps . . . 75
6.10.4 Masks . . . 76
6.10.5 Simple effects . . . 76
6.10.6 Loading and displaying bitmaps . . . 76
6.10.7 Blitting . . . 78
6.10.8 Filters . . . 80
6.10.9 Filtered blits . . . 81
6.10.10 Zooming and scrolling . . . 81
6.10.11 Displaying text and numbers . . . 82
6.10.12 Haikei and bgr functions . . . 83
6.10.13 The graphics stack . . . 83
6.10.14 Controlling screen updates . . . 84
6.11 Animations . . . 84
6.11.1 Basic effects . . . 84
6.11.1.1 Simple shaking . . . 84
6.11.1.2 Layer-based shaking . . . 85
6.11.2 'SerialPdt' animation . . . 86
6.11.2.1 Frame-based animation . . . 86
6.11.2.2 Scrolling animation . . . 87
6.11.3 Videos . . . 88
6.12 Objects . . . 88
6.12.1 Initialising objects . . . 89
6.12.1.1 Text objects . . . 90
6.12.1.2 Number objects . . . 91
6.12.1.3 Environment objects . . . 92
6.12.2 Object management . . . 92
6.12.3 Object position . . . 93
6.12.4 Object attributes . . . 94
6.12.5 Object animations . . . 96
6.13 Timing . . . 97
6.13.1 Waiting . . . 98
6.13.2 Timers . . . 98
6.13.3 Frame counters . . . 99
6.14 System functions . . . 101
6.14.1 Calling extension DLLs . . . 101
6.14.2 Calling external programs . . . 102

6.14.3 Time and date . . . 102
6.14.4 Window settings . . . 102
6.14.5 Saved games . . . 103
6.14.6 System command menu functions . . . 104
6.14.7 Menu mode . . . 105
6.14.8 Skip mode . . . 106
6.14.9 Auto mode . . . 106
6.14.10 CG mode . . . 107
6.14.11 Font settings . . . 108
6.14.12 Miscellaneous flags . . . 108
6.15 Debugging . . . 109

7 RLdev extension libraries 111
7.1 rlBabel: a flexible rendering engine for international text . . . 111
7.1.1 Configuring fonts and international names . . . 112
7.1.2 Dynamic lineation and proportional text . . . 113
7.2 Textout: a compatible rendering engine for Western text . . . 115

8 Gameexe.ini reference 117
8.1 Interpreter configuration . . . 117
8.1.1 Main window settings . . . 117
8.1.2 Locations . . . 117
8.1.3 Bytecode entrypoints . . . 118
8.1.4 Memory settings . . . 119
8.2 Text output . . . 120
8.2.1 General configuration . . . 120
8.2.2 Global font settings . . . 120
8.2.3 Defining a window . . . 121
8.2.3.1 Appearance . . . 121
8.2.3.2 Animations . . . 121
8.2.3.3 Text settings . . . 122
8.2.3.4 Select window settings . . . 123
8.2.3.5 Name window settings . . . 124
8.2.3.6 Local functionality toggles . . . 125
8.2.3.7 Miscellaneous settings . . . 125
8.3 Audio . . . 126
8.4 Debugging . . . 126

A VisualArt's-RLdev name equivalences 128

B The GNU General Public License 132

C The GNU Lesser General Public License 136

D Other licensing information 142

A Very Special Introduction

Hello. I'm Dr. Brian Oblivion.

Acknowledgements

A number of people have been of great help to me in my efforts to understand and document the RealLive system: Jagarl, on whose *scn2kdump* utility, now developed into *xclannad*, I built much of my work; roxfan, whose additional reverse-engineering work has revealed the details of the DLL plugin system and many other built-in functions and functionality; and, not least, the careless programmers of RealLive games themselves, whose consistent and inexplicable failure to remove vast quantities of debugging code from their software on release has been *extremely* enlightening.

My thanks are also due to those users who have contributed in various ways to RLdev's development: Ed Keyes of insani, who tested an earlier version to its limits with *Planetarian*, also generously giving permission for me to incorporate his `g00` compression code into Vaconv; 258-shi, whose introduction of RLdev to a large userbase uncovered numerous bugs; MetaFX, who helped develop the Chinese support, and Arte, who helped with Korean support; and Soulfang, my first user, without whose request the program might never have become anything more than a toy.

Preface

The RLdev package provides tools for manipulating the data used by the VisualArt's RealLive virtual machine.[1]

Since the actual language used by the developers of these games is undocumented, a simple programming language of my own design, called 'Kepago', is used instead. The present implementation is rather basic, but higher-level features will be introduced in future versions. Anyone who has used my older Kpac package (a similar unofficial development kit for the AVG32 virtual machine) will find much here familiar, including many of the function names, although they will doubtless miss the sophistication of that rather more mature Kepago implementation.

RLdev was designed and written to make a *Clannad* translation possible, and extended to support Insani's work on *Planetarian* and my own on *Kanon* when the RealLive edition of the latter was released; it has never been intended to supercede other free visual novel development systems (most of which are easier to use and come with fewer restrictions), and it is certainly not intended to replace the official VisualArt's toolkit, if you're considering licensing the RealLive engine. In short, I hope this toolkit is useful, but I make no apology if the programs have any deficiencies, inaccuracies, inadequacies, or inconsistencies. You have the source code: as the sage advises, "*quicumque melior potest faceret*".

[1] I use the term RealLive to cover the whole current-generation interpreter series from VisualArt's. Version 1.0 was called AVG2000; the name was changed to RealLive around version 1.0.0.8.

Chapter 1

Addendum 2011

Thanks to the generous nature of VisualArt's in 2010, as the decade long lifecycle of its flagship product, RealLive, came to an end, much more is known and knowable about the mysterious source language which Haeleth approximated in a programming dialect of his own creation, known to the world as Kepago.

For the very first time a mountain of documentation, various tools and gizmos, sample code and much more tantalizing tidbits were bundled up into an official Software Development Kit bearing the name RealLiveMax and available to all of Japan from a special web page. But VisualArt's went much further than that. Under the RealLiveMax license there would be no fees whatsoever. Free at last!

Where possible and practical, previously unknown source features, syntax, function names, keyword names et al exposed in the RealLiveMax SDK have been accounted for in RlDev so that standard features detailed in the offical documentation for RealLiveMax will also apply to Kepago++, a hybrid of what was and what will be.

NOTE: It is left as an exercise for the reader to obtain the RealLiveMax SDK and to read and comply with the license for himself if he is so inclined. The anonymous contributor of this section cannot be depended upon as a substitute for your own quest for education and enlightenment.

Your lunch belongs to Zippy.

Chapter 2

Using RLdev

RLdev is split up into utilities according to the types of file each processes. Currently there are four: Kprl to handle RealLive scenario files (see section 5.2), both as archives and separately, Rlc to handle Kepago source code, Vaconv to handle conversions of bitmap files, and RlXml to handle conversions of certain other data formats (as of 1.21, just `GAN` animations).

All utilities have certain things in common: running them without arguments will display a help screen, and the options `--help` (displays the same help screen), `--version` (displays a brief copyright notice), and `-v`, `--verbose` (increases the amount of diagnostic information printed) are always available.

2.1 Kprl: the archiver/disassembler

Synopsis: `kprl` *`options (files | archive [ranges])`*

One option is always required, to select the operation to perform. Which of *`files`* and *`archive`* is used, and what other options (if any) are available, depends on the operation.

Where *`archive`* is used, in all cases other than for the `-a` command, *`ranges`* is a list of integers between 0 and 9,999. Ranges can be specified, e.g. '`414-416`'. The files actually processed will be the intersection of the set specified on the command line and the set of files present in the archive. Omitting *`ranges`* causes all files to be processed.

Common operations

`-a, --add`
Updates files in *`archive`*; if the files listed are not present, they will be added, and if the archive does not exist, it will be created. *`ranges`* should be one or more filenames, which are taken to be the names of scenarios to add. Their names must begin `seen`*N*, where *N* is an integer between 0 and 9,999.

`-k, --delete`
Removes the files indicated by *`ranges`* from *`archive`*. If all files are removed, the archive will be deleted; to prevent accidents, *`ranges`* may not be omitted.

-l, --list
Lists files in *archive*, printing names, sizes, and compression ratio where applicable. The files listed can be restricted with *ranges*.

Additional options:

-N, --names
Instead of sizes, print a list of characters appearing in the scenario (not available in AVG2000).

-d, --disassemble
Disassembles files, producing Kepago source code corresponding to their RealLive bytecode. If an *archive* and *ranges* are given, files will be extracted from the archive automatically; otherwise *files* will be treated as a list of separate scenarios to be disassembled.

Additional options:

-o *DIR*, --outdir=*DIR*
Places output in *DIR*

-e *ENC*, --encoding=*ENC*
Selects a character encoding for output. Valid options always include `cp932`[1], `euc-jp`, and `utf-8`. The default is normally `cp932`, but this may be configured differently depending on the settings used to build RLdev.

--bom
If the output encoding is UTF-8, causes a BOM (byte-order mark) to be prepended to all output files. The default is to leave the BOM out, but some programs (notably Microsoft's text editors) can not edit UTF-8 files without it.

-s, --single-file
By default, Kprl puts text strings into a separate resource file; this makes it much easier to translate text without being distracted by implementation details. If this option is supplied, all strings are included in the source code instead.

-S, --separate-all
Normally the only strings placed in the resource file are those which Kprl can identify as part of the game's script. If this option is supplied, all strings containing Japanese text are separated. This usually catches any parts of the game's script that the default mode might miss, but it can also clutter up resource files with false positives. It obviously cannot be used with `-s`.

-u, --unreferenced
By default, all code is disassembled, even where it can be proven that it is never executed (for example, code immediately following a goto or end()). Enabling this option will suppress any code that is provably unreferenced; this is normally safe.

-n, --annotate
Adds copious comments to the generated code. Two types of comment are created: offsets in the bytecode to each command are noted, and some function parameters are given labels that roughly indicate their meaning.

[1] CP932 is Microsoft's version of Shift_JIS; this is RealLive's native character encoding.

-r, --no-codes
By default, a number of functions which primarily affect text display are disassembled as control codes within string literals. Enabling this option will disable this behaviour.

-g, --debug-info
The official VisualArt's RealLive compiler puts a large amount of strictly unnecessary data in the bytecode for debugging purposes. By default it is ignored, but this flag causes Kprl to read it and include information from it in the source code generated.[2]

-t *TARGET*, --target=*TARGET*
Specifies the format of the data to disassemble (one of `reallive`, `avg2k`, or `kinetic`). By default, Kprl detects a format automatically. It is normally only necessary to use this option when disassembling code from a Kinetic Novel that uses `kinetic.exe`, since the `kinetic` format is subtly different from the `reallive` format but is always detected as the latter.

-f *VER*, --target-version=*VER*
Specifies the version of the bytecode to disassemble (either as a version number consisting of up to four integers separated by dots, or as the name of an interpreter executable to query for a version number). By default, Kprl detects a version automatically.

Less common operations

-x, --extract
Decompresses and decrypts files. As with `-d`, `-x` can process archives or standalone scenarios. The suffix `.uncompressed` is added to all file extensions, to prevent inadvertent clobbering.

Additional options:

-o *DIR*, --outdir=*DIR*
Places output in *DIR*

-b, --break
Extracts files from *archive*, without decompressing them.

Additional options:

-o *DIR*, --outdir=*DIR*
Places output in *DIR*

-c, --compress
Compresses and encrypts files. *files* must be a list of uncompressed scenarios. It is normally not necessary to compress files manually; Rlc compresses the bytecode it generates by default, and Kprl compresses files if necessary when archiving them. When compressing, Kprl removes the suffix `.uncompressed` from the file extension, if present.

[2] In practice this just means the source will be cluttered up with `#line` directives giving the lineation of the original code from which the game was compiled, which is probably not incredibly useful.

2.2 Rlc: the compiler

Synopsis: `rlc` *[options]* *file*

file is the name of a Kepago source file to compile. Only one main file can be processed at a time, though it may include code from other files. It will be compiled to RealLive bytecode in a standalone scenario, which can either be interpreted directly or added to a scenario archive.

The following options are recognised:

Files and Directories

-o *FILE*, --output=*FILE*
Sets name of output file to *FILE*. If no name is specified on the command line, Rlc will look for a `#file` directive in the source code; if none is found, the output file will be called `rlas_output.txt`.

-d *DIR*, --outdir=*DIR*
Places output file in *DIR*.

-r *DIR*, --resdir=*DIR*
Specifies a directory in which resource files are located. If provided, *DIR* will be prepended to the filename provided by a resource directive.

-i *FILE*, --ini=*FILE*
Specifies a `gameexe.ini` to use, or the directory containing one. Access to the `gameexe.ini` that will be used at runtime is required in order to read various configuration data. If none is specified on the command line, Rlc will look for an environment variable `GAMEEXE`; failing that, it will look in the directory containing the source file being compiled.

Text Encoding and Transformation

-e *ENC*, --encoding=*ENC*
Selects a character encoding for the input. The options and default are the same as in the disassembler (see above).

-x *ENC*, --transform-output=*ENC*
Selects a character encoding transformation for the output. See 4.5.2 for details.

- *BOOL*, --force-transform=*BOOL*
If true, warning messages are issued when input text cannot be represented using the encoding specified for the scenario generated in the output. If false, transformation failures result in an error message is displayed and compilation is aborted. Non-fatal warnings are issued by default.

-t *TARGET*, --target=*TARGET*
Specifies the output target (one of `reallive`, `avg2k`, or `kinetic`).

By default, Rlc attempts to detect the target automatically by looking for an interpreter executable in the directory containing the `gameexe.ini` being used; failing that, it falls back on RealLive as the most likely option. The `#target` directive can also be used to set a different target. This option takes precedence over the directive, which in turn takes precedence over auto-detected targets.

-f *VER*, --target-version=*VER*
Specifies the version of RealLive bytecode to compile for; see section 5.1.1 for the distinction between target and version. *VER* may be either a version number (between one and four integers separated by dots) or the path to an interpreter executable which will be queried for its version.

By default, Rlc attempts to detect a suitable version automatically by looking for an interpreter executable in the same way as for the `--target` option and querying its version number. If this fails, it defaults to version 1.0 (if the target is AVG2000) or 1.2.6 (for other targets). You can always use the `#version` directive to override this; unlike the `#target` directive, `#version` also overrides explicit version specifications.

Low-level Compiler Options

-u, --uncompressed
Disables automatic compression and encryption of output. The output will be given a `.uncompressed` suffix, as though it had been compiled normally and then decompressed with `kprl -x`. This is only useful if you need to examine the generated bytecode.

-g, --no-debug
Strips debugging information and calls to debugging functions from the compiled bytecode. The output will be smaller and marginally faster.

--no-assert
Disables assertions (see `assert()`).

--safe-arrays
Enables runtime bounds checking for Kepago arrays. This option has no effect if `--no-assert` has also been specified.

--flag-labels
This feature is experimental and incomplete.

Enables automatic generation of the `flag.ini` symbol information file used by the RealLive debugger. Variables declared with the `labelled` directive will be added to `flag.ini` automatically.

Currently, `flag.ini` is created automatically in the same directory as the `gameexe.ini` in use; any existing `flag.ini` will be deleted automatically without confirmation, and you will have to make your own arrangements if you wish to combine labels from more than one source file or to make use of `flag.ini` features other than basic variable labels. The generated labels consist of the variable's name (and, in the case of arrays, its length). All labels are placed in flag group 0.

Game-specific Options

-G *GID*, --game=*GID*
Specify the Rlcgame using an id which can be used to look up game specific information that may be useful, if not critical, for producing compatible bytecode and handling those special cases where custom encryption keys are employed to keep your precious bytecode data *in* and those armies of perverts who want to put their hands on your privates *out*.

To this end, the modern Rlcdeveloper is armed with a potent `game.info` file, a private groping hentai's worst nightmare.

`-c` *LONG*, `--compiler=`*LONG*
Provide a somewhat cryptic compiler version number (eg 10002, 110002, etc) as a long integer. Or pass a game id from `game.info` using `-G` instead.

`-k` *KEY*, `--key=`*KEY*
Provide a hex string representing a game specific encryption key to further obfuscate the generated scenario data. Or pass a game id from `game.info` using `-G` instead.

Obsessive-compulsive Minutae

`-F` *LINE*, `--from=`*LINE*
Indicate the line number within the script file at which to begin compilation.

`-T` *LINE*, `--to=`*LINE*
Enumerate a line number within the script file at which compilation will cease.

`-` *FILE*, `--kfn=`*FILE*
Demand that the compiler use a specific RealLive Function Definition File (the default is lib/reallive.kfn.

`-` *EXT*, `--ext=`*EXT*
Decree that script files to be compiled to bytecode will include have the specified file extension (default is 'org' in order to make use of the source code debugger available in debug mode).

2.3 Vaconv: the graphic converter

Synopsis: `vaconv` *[options] files*

files is the names of the bitmap files to convert.
The following options are recognised:

-d *[DIR]*, --outdir=*[DIR]*
Places output files in *DIR*. Passing this option without specifying *DIR*, or giving it the special value PRESERVE, causes output files to be placed in the same directory as their respective inputs. If this option is not passed at all, output files are placed in the current working directory.

-o *FILE*, --output=*FILE*
Sets name of output file to *FILE*. The default is the base name of the input file with an extension selected according to the target format. Note that this option can only be used when only one input file is supplied.

-f *FMT*, --format=*FMT*
Sets output format to *FMT*. Possible formats are PDT, G00, and PNG.

The default behaviour depends on the extension of the first input file: if it is `.g00`, the default output format is PNG, otherwise it is G00.

-g *FMT*, --g00=*FMT*
Allows you to select the G00 format to use (0, 1, or 2); see 2.3.1. The default behaviour is to select an appropriate format automatically based on the data.

This option implies –f G00; you should not use `-f` and `-g` together.

-i *FMT*, --input-format=*FMT*
If Vaconv is failing to recognise the format of the source file, you can bypass the auto-detection by specifying the format manually. Legal values of *FMT* are the same as for `-f`.

-m *FILE*, --metadata=*FILE*
For G00 output, allows you to supply a metadata file to specify features like regions; see 2.3.2. If no metadata file is specified, Vaconv looks for one accompanying the source file with the same base name and the extension `.xml`. If this is not found, default values are used.

For G00 input, allows you to override the name to be used if a metadata file is generated.

As with `-o`, this option can only be used when only one input file is supplied.

-q, --no-metadata
For G00 output, causes Vaconv to ignore all metadata. For G00 input, disables metadata output even where this would lead to loss of information.

-b, --best
Maximise compression. Where this has any effect, it runs significantly slower (by an order of magnitude in the worst cases) than the default algorithm; for G00 output in format 0, however, where the regular algorithm is particularly inefficient, I have seen file sizes almost halved, so it may be worth using for releases.

2.3.1 Bitmap formats

RealLive makes use of two proprietary bitmap formats: PDT and G00.

The PDT format is a holdover from the older AVG32 system, and is the standard bitmap format in AVG2000: it stores 8-bit paletted or 24-bit RGB bitmaps, together with an optional 8-bit alpha mask. In RealLive proper it is normally used only for cursor images.[3]

G00 is the true native format, used for most graphics in RealLive games. There are three G00 formats, each (naturally) with their own advantages and disadvantages:

- **0** 24-bit RGB, no mask
- **1** 8-bit paletted; palette is ARGB
- **2** 32-bit ARGB, plus extra features (see below)

Formats 0 and 1 are usually the most efficient, depending on the type of data; format 2 is the most flexible. By default, Vaconv attempts to select the most appropriate format automatically when converting to G00.

2.3.2 Advanced G00 features

Format 2 G00 bitmaps can contain multiple sub-bitmaps, termed "regions". These are used in some object functions (see 6.12) to create objects with multiple states, such as buttons; such functions generally have a parameter `pattern` which references a G00 region.

When creating a G00 bitmap, regions are defined by supplying an XML metadata file along with the source image. The format is very simple, and an example should be sufficient documentation. This is for one of the CG mode thumbnails in *Clannad*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vas_g00 SYSTEM "vas_g00.dtd">
<vas_g00 format="2">
  <regions>
    <region x1="0" y1="0" x2="114" y2="86" />
    <region x1="0" y1="0" x2="114" y2="86" />
    <region x1="0" y1="87" x2="114" y2="173"/>
    <region x1="0" y1="87" x2="114" y2="173"/>
    <region x1="0" y1="0" x2="114" y2="86" />
  </regions>
</vas_g00>
```

Here five selectable regions are defined, but only two real regions. These are selectable in-game with commands such as

```
objOfFile (0, 'SCGMA00') // Initialises object 0.
objPattNo (0, 1)         // Selects region 1 (this is the same as 0, so has
                         // no visible effect ).
objPattNo (0, 2)         // Selects region 2 (switches to display the other
                         // sub-image).
```

[3] Vaconv's implementation can read all PDT images, but it cannot write paletted PDTs. The paletted format is extremely rare, however.

2.4 RlXml: the auxiliary data converter

Synopsis: `rlxml` *[options] files*

files is the names of the files to convert. Their format will be determined automatically from their extension. If they are RealLive data files, they will be converted to a human-readable XML format; if they are XML files, they will be converted back to the appropriate RealLive binary format.

Currently only two formats are recognised: `GAN` animation files (extension `.gan`), and their corresponding XML versions (extension `.ganxml`).

The following options are recognised:

`-o` *NAME*, `--output=`*NAME*

The meaning of this option depends on the number of files being converted. If only one is being converted at a time, then `-o` specifies the path and name of the converted file; if multiple files are being converted at once, `-o` specifies the directory in which the converted files should be placed, and their names will be derived automatically from the input files' names.

If the option is not given at all, then output is placed in the current working directory with automatically-generated names.

Chapter 3

Getting started

Ease of use is a function of time, and I haven't had that much time to feed into it, so you'll have to do a bit of work to start a project with RLdev. Moreover, what's required will change (hopefully for the better) over time, so be sure to check this section again with each new release.

3.0.1 The project header file

If there is a file called `global.kh` in the same directory as a source file, Rlc will automatically include it, as though every source file began with the line `#load'global'`. Placing configuration options and common definitions in such a file will automatically share them among every source file in the directory.

You can of course use a different name for your project header, but if you do, you will have to load it manually.

Overriding or extending the project header on a script by script basis may be useful as well. Auto inclusion of a script header file will occur immediately after the project header where a corresponding global header `global****.kh` can be found in the same directory as the script source file sharing the same four digit index.

Sample Output:

Loading project header 'global.kh' Loading script header 'global0010.kh' Compiling script file 'seen0010.org'

3.0.2 Memory allocation

You must identify two sections of memory for Rlc to use to allocate temporary variables. At a minimum you must provide a block of 100 local integers and a block of 10 local strings.

By default, Rlc uses the whole C[] array for temporary integers, and S[1900] to S[1999] for temporary strings. If you are writing a program from scratch, you can either avoid using these memory areas manually, or change them to something more convenient. If you are modifying an existing program, you will have to analyse its memory usage yourself to discover whether these defaults are suitable, and modify them if they are not. Be warned that any direct access or modification of a variable in a space that has been assigned to Rlc could cause your program to malfunction, but Rlc has no way to detect such conflicts automatically.

To modify the settings for memory allocation, call the `rlcSetAllocation()` function. You must call this function at the start of *every* file in your project — it is evaluated at compile-time, so it is not enough simply to call it at the start of the first scenario. The easiest thing is just to place it in your `global.kh` (see above), which is automatically included at the start of every scenario.

Chapter 4

Kepago

Kepago is a simple imperative programming language in the style of C or Pascal. It was designed as a substitute for the unknown language used by VisualArt's themselves. The reference implementation remains my `kpc` compiler for AVG32.

The present implementation of Kepago for RealLive is limited in scope; it currently lacks major features such as user-defined functions, which limits its expressivity considerably. It is, however, adequately usable for simple programming tasks.

4.1 Lexical structure

The encoding of a file is determined by the default encoding for which Rlc was compiled (this is CP932 in the standard configuration), and can be overridden on a case-by-case basis on the command line.

Identifiers can use almost any character defined in the current encoding. They are not case-sensitive. They may not begin with numbers, or with the reserved characters $ and @. Identifiers beginning and/or ending with two underscores are reserved for internal use: they are valid, but you should avoid using them except for the specific cases documented in this manual.

Numbers are decimal by default; digits can be separated with underscores, as in ML. Hexadecimal numbers are identified by a prefix $, as in Pascal. Binary numbers take the prefix $#, and octal numbers $%.

Comments come in two flavours. Line comments begin with `//`, and block comments (which behave like C comments, i.e. they cannot be nested) are delimited by `{-` and `-}`.

4.2 Structure of a file

A Kepago file is simply a text file containing a sequence of statements and definitions. There is no statement separator: statements are neither line-delimited like Basic nor semicolon-delimited like C and Pascal, although line breaks and commas can be used almost arbitrarily to separate them for clarity.

A file can be ended with the **eof** keyword; nothing following this will be processed by Rlc.

4.3 Expressions

Expressions are fairly ordinary. The following binary operators are recognised. Their meanings are identical to C, but their precedence is not:

```
<< >>
*  /  %  &    // higher precedence
+  -  |  ^
>= >  <= <
== !=
&&             // lower precedence
||
```

In addition, the operators −~! are recognised as unary operators, again with the same semantics as in C; these all have equal precedence greater than any binary operation. Precedence can, as usual, be overridden with parentheses.

The arithmetic operators (from | upwards) can have an = appended to form assignment operators, again as in C; note however that these cannot be used on the right-hand side of an expression or within function calls. There are currently no unary increment/decrement operators and no address or pointer dereference operators.

Example:

```
x = 1
x += (strlen(s) + 2) * 3
```

When processing strings, the only permitted operator is +, which performs string concatenation. It is also legal, within integer expressions, to compare strings with the comparison operators. Apart from that it is not legal to mix strings and integers in expressions, though the various integer types may be mixed freely.

A number of Kepago constructs utilise the concept of a “constant expression”. This refers to any expression that can trivially be fully evaluated at compile-time. A constant expression may contain the full range of operations, and it may refer to constant symbols, but not variables. It may also contain calls to certain functions, such as `max()`, that can be evaluated at compile-time, and to a set of macros, such as `defined?()`, that are always evaluated at compile-time.

4.4 Statements

In addition to the statement types detailed in this section, it is also valid to use a string expression by itself as a statement, which is how text display is normally accomplished. This topic is discussed in depth in section 4.5.

4.4.1 Blocks and scopes

Anywhere a statement is valid, it can be replaced with a block. Blocks are opened with : and closed with ;, and function in much the same way as braces in C-style languages.

Statement blocks double as scoping constructors. Symbols are visible only within the scope in which they are defined, or scopes nested within it. For example:

```
int foo = 1
if (foo == 1):
  int bar = 2
  'foo is \i{foo} and bar is \i{bar}' // prints "foo is 1 and bar is 2"
  pause;
'foo is \i{foo}' // prints "foo is 1"
'bar is \i{bar}' // compile–time error: bar is not visible in the outer scope
pause
```

4.4.2 Labels

While higher-level control structures should be used where possible, program flow may also be controlled with `goto`, `gosub`, and related functions. Labels for use with goto statements can be defined at any point where a statement is valid; a label is any valid identifier beginning with @. The scope of a label is global; if the same label is defined multiple times, a warning is issued and the later definition is used.

Example:

```
@loop
  'This text is displayed repeatedly in an infinite loop.'
  pause
  goto @loop
```

You should, of course, write this instead:

```
while 1:
  'This text is displayed repeatedly in an infinite loop.'
  pause;
```

Rlc optimises conditional tests away when the conditions can be evaluated at compile-time, so it generates identical code in both cases.

4.4.3 Variable declarations

Declaration syntax

Declaration statements are of the form
type [(*directives*)] *variables*
type is the type given to all the variables declared by this statement, *directives* is an optional parenthesised list of flags that control their declaration, and *variables* is a comma-separated list of declarations.

Each declaration in *variables* consists of an identifier which may optionally be followed by an array declaration (see next section), an initial value, and/or an address specifier. Initial values are declared with the form *identifier* = *value*; addresses are declared with the form *identifier* –> *space.address*. For example:

```
byte a                    // value undefined, address automatic
byte b = 2                // initialised to 2, address automatic
byte c –> MEMARR_A.4000 // value undefined, address A8b[4000]
byte d = 2 –> MEMARR_Z.10 // initialised to 2, address Z8b[10]
```

The types currently supported are:

`int`	32-bit integer
`byte`	8-bit integer
`bit4`	4-bit integer
`bit2`	2-bit integer
`bit`	1-bit integer
`str`	character string

The directives currently supported are:

`block`

Ensures that all variables declared in the one statement are allocated contiguously, and, in the case of integers smaller than 32 bits, ensures that they are packed.

An example will illustrate the use of this:

```
int x, y, z
sum (x, z)          // value is undefined
int(block) x, y, z  // x, y, and z guaranteed to be contiguous
sum (x, z)          // guaranteed to equal x + y + z
```

`zero`

Ensure that the allocated memory is initialised. Integers not otherwise given initial values will be set to 0; strings will be set to the empty string.

`labelled`

If the `--flag-labels` option was passed to Rlc, entries will be created in `flag.ini` for variables declared with this directive.

`ext`

Variables declared `ext` are not constrained by enclosing blocks; they remain in scope permanently from the point of their declaration unless deallocated manually.

Arrays

Arrays are allocated in the same way as other variable declarations, with the addition of an array declaration immediately following the variable's identifier. This consists of the standard square brackets surrounding an integer constant giving the array's length.

An array can be initialised in any of three ways. Firstly, if the `zero` directive is included in the declaration, the memory allocated will be cleared automatically. Secondly, if a single expression is given as an initial value, every member of the array will be initialised to that value. Finally, individual members of the array can be initialised by providing a tuple as the initial value:

```
int a[5] = { 1, 2, 3, 4, 5 }
```

In this last case, and only in this case, the square brackets may be left empty, and the length of the array will be set automatically to the number of elements provided:

```
str strings[] = { 'foo','bar','baz' }
```

Only one-dimensional arrays are supported.

Caveats

Memory allocation is currently handled statically based on the settings passed to `rlcSetAllocation()`.

All blocks allocated are word-aligned: that is to say, bit x, y, z allocates three whole words even though only three bits are being used. You can get round this by allocating variables of smaller sizes in arrays or with the `block` directive.

Be aware that the variable allocation system is *not sound*.

```
if 1:
  int x = 0
  gosub @oops
  'x = \i{x}'
  pause
;                 // end of scope, x implicitly deallocated
end

@oops
  int y = 1       // y allocated at the same address as x
  ret
```

prints x = 1, not x = 0, even though the value of the logical variable x was never modified! Safe usage of the current system requires that any variables be allocated outside the largest scope in which they are used; in the example above, x would have to be declared at the top level, and any variable which is to be used after a call to another scenario should be declared globally in a project header file to ensure that its memory will not be used for anything else unintentionally.

A future version of Rlc will hopefully introduce dynamic memory allocation, which will mitigate these problems to some extent.

4.4.4 Constant declarations

"Constant" implies that the value is always known to the compiler, not that the value is immutable; Kepago constants are actually more like compile-time variables than constants in the strict sense.

Constants can hold integer or string values, and can be used in expressions wherever a literal would be valid.

They are handled with the following directives (see also 4.4.6).

`#define IDENT`

Defines the symbol *IDENT*. Multiple symbols can be defined at once, separated by commas. The value bound to *IDENT* is a non-zero integer.

The scope of a symbol defined with this directive is not constrained by blocks; it remains defined in all code that is *compiled* subsequent to its definition. In the current state of the implementation, this effectively means that it remains defined to the end of the file. Once user-defined functions are implemented things will become a little more complicated.

`#undef IDENTS`

Undefines the given symbol(s).

At present, the only symbols which may be undefined manually are those which were defined globally with `#define`.

`#const` *IDENT* = *EXPR*

Defines a new constant symbol. The constant expression *EXPR* (see 4.3) is evaluated, and its value assigned to *IDENT*.

Multiple symbols can be defined at once, separated by commas.

The scope of a symbol defined with this directive is limited to the current block.

`#set` *IDENT* = *EXPR*

Mutates the value assigned to the symbol *IDENT*. This does not affect the symbol's scope.

As a shortcut, you can use the form `#set` foo += 1 instead of `#set` foo = foo + 1; the same goes for other arithmetic operators.

4.4.5 Inlines/Macros

In C, `#define` statements can be used to specify function like macros which can accept multiple parameters and which are replaced with a macro's result when expanded by the C preprocessor.

Similar syntactic magic can be conjured up within a kepago file by making use of an inline macro definition. Note however that while it's convenient to imagine that C and kepago macros are equivalent, the syntax used to define such macros diverge, so it might be useful to dismiss such convenient fantasies and read on.

#inline *IDENT*(<arg1>,<arg2> = *EXPR*[<arg2>] ...) : *STMTS* ;

Defines a macro which can be used (as a macro call) within expressions. The macro call is expanded during compilation and replaced inline with the resulting constant expression. As the macro call vanishes entirely during the compilation phase it would be a mistake to confuse an inline macro call with an actual function call which generates bytecode that performs an actual call which returns a value upon completion. So don't be a fool!

It might also be considered lazy thinking to conflate so-called intrinsic functions with inline macros. While both perform feats of shapeshifting when encountered by the rlc compiler, intrinsic functions are predefined within the compiler itself rather than being defined in terms of kepago statements as inline macros are.

User-defined functions remain a potential feature to possibly be implemented in some alternate future version whose existence is only found in the shady realm of urban legends between fits of delerium tremens.

If this has failed to clarify the situation and confusion prevails, a round of hands on coding experience and experimentation may lie in your immediate future. May the wistful spirit of kepago be with you as you go forward into the unknown.

There are plenty of inline macros to explore in library files bearing the .kh extension. This might be a wise place to fortify your kepago skills.

4.4.6 Directives

Like in C, directives (that is, statements which modify compiler state rather than generating code) begin with a hash sign #; unlike in C, these are processed by the

compiler itself, not a preprocessor.

The following directives are recognised in the current version of Rlc:

#load`'FILE'`

Loads a header file. Rlc looks first for *FILE*, then for *FILE*`.kh`, first in the same directory as the source file, and then in the RLdev library directory. The contents of the file are parsed at the current position in the same way as C's #include directive.

#file`'FILE'`

Sets a default output filename. This can be overridden on the Rlc command line with the `-o` option.

#target TARGET

Selects a default target; *TARGET* should be `RealLive`, `AVG2000`, or `Kinetic`. This can be overridden on the Rlc command line with the `-t` option. Note that the parameter is an identifier, not a string.

#version *A[.B[.C[.D]]]*

Selects a bytecode version to generate code for; this is the equivalent of the `-f` command-line option, and functions identically.

#resource`'FILE'`

Loads the named resource file, providing access to all the strings it contains. The scope of its definitions extends from the *#resource* directive to the end of the file; it must precede any *#res* directives referencing its contents.

#base_res `'FILE'`

Loads the named resource file, providing access to all the strings it contains for use as a base of resources in a multi-version translation project. The scope of its definitions are limited to a resource file previously loaded using a *#resource* directive. (New feature for version 1.45.)

#res<*ID*>

Inserts the resource string with the identifier *ID* at the current position. See section 4.6 for details.

#entrypoint INDEX

Places an entrypoint at the current location in the code; entrypoints function as labels for the `jump()` and `farcall()` functions to jump into a scenario. *INDEX* should be an integer between 0 and 99 inclusive.

If *INDEX* is 0, the directive is ignored. Entrypoint 0 always comes at the very start of the scenario, and Rlc defines it automatically. Superfluous entrypoint directives are accepted for compaitibility reasons.

#character`'NAME'`

Adds *NAME* to the *dramatis personae* table in the header of a bytecode file. This is what `kprl -lN` reads.

Character name data appears not actually to be used by RealLive; its purpose is a mystery, but it's probably just debug information. It does not exist in AVG2000, so the `#character` directive is ignored when compiling for that target.

`#line` *LINE*

Tells Rlc to start counting lines from the given value, rather than the actual line position in the source file. May be of use if you want to preprocess code or use literate programming tools, but it's really only there to give the disassembler something to do when asked to read debug information.

`#kidoku_type` *TYPE*

Determines the format to use for kidoku markers in generated bytecode. If the executable is to be run with RealLive 1.2.6.6 or later, this should usually be 2, although it appears not to be critical that this be the case; in all other cases it *must* be left with its default value of 1.

Note that this is a global setting; the value that will be used for a given output file is that assigned most recently when the end of the file is reached.

`#print` *EXPR*

Causes the compiler to print the current value of *EXPR* to stderr. *EXPR* must be a constant expression, but it can be an integer or a string.

`#warn` *EXPR*

As `#print`, but the output is formatted as a compiler warning.

`#error` *EXPR*

As `#print`, but the output is formatted as a compiler error, and compilation is halted.

4.4.7 Conditional compilation

The code to be compiled can be selected at compile-time by using a further set of directives. As with the others, note that these are *not* related to any sort of preprocessor. They are block statements: they can only be used where a statement would be valid, and the code they enclose is analysed for syntactic validity even if it is not compiled.

`#if` *EXPR*

`#elseif` *EXPR*
`#else`
`#endif`

Select a block of code to compile based on the value of *EXPR*:

```
#const First = gameexe('SEEN_START')
#if First == 1
  #print 'The game begins with scenario 1'
  // Code for this case
```

```
#elseif First == 2
  #print 'The game begins with scenario 2'
  // Code for this case
#else
  #error "The game begins with scenario \i{First}, and I don't \
        know what to do"
#endif
```

Unlike :; blocks, the contents of these directives are not counted as blocks for scoping purposes.

#ifdef EXPR

#ifndef EXPR

It is often convenient to be able to determine whether a given symbol has or has not been defined. These two directives are shorthand ways to do this: they are exactly equivalent to *#if*, except that every identifier in *EXPR* that is not directly compared with anything else is replaced with a call to the `defined?()` macro.

```
#ifndef foo && bar && baz > 1 // -> #if !defined?(foo) && !defined?(bar) && baz > 1
  {- do stuff -}
#endif
```

4.4.8 Control structures

if *CONDITION STATEMENT*
if *CONDITION STATEMENT* **else** *STATEMENT*

A standard conditional control structure. *CONDITION* is evaluated; if the result is non-zero, *STATEMENT* is executed. The optional **else** allows a statement to be specified for execution if the condition evaluates to zero.

Kepago adopts the usual rule to resolve the 'dangling **else**' problem: each **else** is taken to apply to the innermost available **if**. This can be overridden by use of blocks:

```
if rnd(10) < 5:
  if SyscomEnabled, 'your code here';
else
  'Without the :; above, this would attach to \
   the inner `if\'.'
```

while *CONDITION STATEMENT*

If the value of *CONDITION* is non-zero, *STATEMENT* is executed repeatedly; *CONDITION* is checked after each iteration, and the loop exits when it evaluates to zero.

You can exit the loop prematurely with the **break** statement, or jump straight to the next iteration with the **continue** statement.

repeat *STATEMENTS* **till** *CONDITION*

The opposite of **while**. *STATEMENTS* is executed at least once, and the loop continues for as long as *CONDITION* is zero.

For historical reasons, and unlike the other structures in this section, the **repeat** loop is a block in itself: you can include multiple statements without a :; block.

for (*INITIALISATION*) (*CONDITION*) (*INCREMENT*) *STATEMENT*

A C-style 'for' loop.

```
for (int i = 0) (i < 10) (i += 1):
  do_stuff(i);
```

is equivalent to

```
: int i = 0
  while i < 10:
    do_stuff(i)
    i += 1;;
```

case *EXPR*
of *CASE*
other
ecase

EXPR is compared to each *CASE* in turn; if a match is found, the following code is executed. The **other** clause is equivalent to an **of** clause, but always matches; it is optional, and if present must be the last clause.

As with C's 'switch' statement, each case should be terminated with a **break** statement, or execution will fall through to the next case. This is useful where multiple values require the same treatment:

```
int qual = SoundQuality
'Sound mode is '
if qual % 2, '16-bit ' else '8-bit '
case qual
of 0
of 1
  '11 kHz'
  break
of 2
of 3
  '22 kHz'
  break
of 4
of 5
  '44 kHz'
  break
other
  '48 kHz'
ecase
pause
```

Rlc will attempt to optimise the test and jumps away if *EXPR* can be evaluated at compile-time. In certain cases it may be desirable to know whether this optimisation has succeeded (for example, when writing **other** clauses that raise an error, you may wish to raise the error at compile-time in such cases). Rlc therefore defines the symbol __ConstantCase__ when compiling **case** blocks; if it is non-zero, then the innermost case block currently being compiled has been optimised away.

4.4.9 Function calls

With one or two exceptions, function calls use the standard C-like syntax: the function name followed by a list of parameters, bracketed and comma-delimited.

Anything valid on the right-hand side of an assignment expression is also valid as a function parameter. Certain functions permit other constructs, such as tuples (parameters containing multiple expressions; in Kepago these are enclosed in curly braces), and one or two have wholly different syntaxes. All these exceptions are documented under the appropriate function in chapter 6.

Unknown functions

It may sometimes be desirable to make use of a function that exists in RealLive but is not currently exposed through the Kepago/RealLive API. There exists a 'raw opcode' command that can be used to insert such functions: it has the format '**op**<*type*:*module*:*opcode*, *overload*>', which can be followed with a parenthesised parameter list as any function (but cannot be used in assignments). The variables are all integers; *type* and *module* appear to be used to identify groups of related functions, *opcode* identifies a particular function (its value is only unique within a particular type/module combination), and finally *overload* identifies an instance of that opcode (different instances have the same broad semantics, but take varying numbers of parameters).

Mnemonic aliases are defined for some values of *module*, in the hope that this may give some hint as to the purpose of an unknown function when Kprl disassembles a scenario containing it. For example, the `pause` function can also be invoked with **op**<0:Msg:00017, 0>.

In the general case, however, it is preferable to add support for the function to the API than to use it with **op**<>. This can be accomplished by editing `reallive.kfn`, or more easily by providing the author with an example and description, upon which he'll happily do it for you.

4.5 String handling

Strings can be delimited with either `'` or `"`; there is no semantic difference between the two. Due to the limitations of the RealLive system itself, only JIS characters are valid in strings. To use within a string a quote of the type being used to delimit that string, it must be escaped with a backslash; line breaks must also be escaped, and any whitespace at the start of the following line will be ignored, unless this too is escaped:

```
str s = "This string's value \
      is a single line, spaced\
    \ normally.")
```

Note however that arbitrary characters may not be escaped: any escaped alphabetic character is taken to open a control code, as will an escaped left brace.

4.5.1 Displaying text

Display strings (that is, strings used directly as statements, rather than in expressions) are passed directly to RealLive's text display routines. When one is encountered, a text window of the currently active type is opened and the text printed in

it. (Text window types are defined in `gameexe.ini`, as described in 8.2.3, and selected with the `TextWindow()` function.)

4.5.2 Usable characters

The RealLive bytecode format requires all textual data to be in the Shift_JIS encoding. By default, therefore, RLdev converts text to Shift_JIS: it follows that the only characters which are valid in Kepago strings are those in the JIS X 0201 or JIS X 0208 character sets. This is perfect for Japanese text, and sufficient for most English.

Other languages require characters not present in the JIS character sets. There are two solutions to this issue. One is to fake the required characters: some glyphs can be included as bitmaps with `\em`, others (such as accented letters) can be built up through overprinting by using `\mv`. A better solution — in some cases, such as Chinese, the *only* solution — is to use a non-JIS character set, with a non-standard encoding that has the same characteristics as Shift_JIS, and to arrange for this to be decoded on the fly in an intermediate layer between RealLive and the operating system.

In RLdev, this is accomplished in three stages:

First, the UTF-8 encoding should be used for source code files. This is the only supported input encoding that can handle non-Japanese text. If it is not the default encoding for your copy of RLdev, you will have to specify the option `"-e utf8"` when compiling the code.

Second, an *output encoding transformation* is applied. This is done by passing the "`-x` *ENC*" option when compiling the code, where *ENC* is the name of a supported transformation (see below).

The final step is to arrange for the RealLive interpreter to be able to decode the transformed text. This can be done either by modifying the interpreter itself, or by using an extension DLL to hook into it at runtime. A suitable DLL is provided in the form of the rlBabel library (see 7.1).

RLdev currently supports the following transformations:

None
: For Japanese or ASCII text. This is the default.

Chinese
: For Simplified Chinese text. The output uses a non-standard encoding of the GB2312 character set (based on a system formerly used by the Key Fans Club).

Korean
: For Korean text. The output uses a non-standard encoding and character set. The characters encoded comprise the non-hanja parts of KS X 1001, plus the 4000-odd additional precomposed hangul from the ill-fated Unicode 1.1: this appears to be the simplest practical compromise for encoding modern Korean text, though the encoding could potentially be expanded further if necessary.

Western
: For Western European text. The output uses a non-standard encoding of the Windows CP1252 character set (ISO-8859-1 with extensions).

  Since this encoding uses double-byte characters to represent some half-width characters, it should be used with the rlBabel lineation library (7.1.2) to ensure correct character spacing.

4.5.3 Control codes

Control code syntax is TEX-style, *identifier*{}, where what goes between the curly braces has the same syntax as the contents of the parentheses in a normal function call — in most cases the braces are either empty or contain a single integer or variable. Note that for control codes of more than one letter, braces are always required, even if no parameters are being passed to the code.

Descriptions of the basic control codes currently available follow. In addition to these, there are also more complex codes \name and \g, described in the following subsections, and a set of control codes only valid in resource files, described in section 4.6.

\n

Forces a line break at the current position, retaining indentation.

\r

Forces a line break at the current position, resetting indentation.

\p

Inserts a pause. Text display will halt until the player clicks to advance, but it will then continue where it left off, without clearing the screen.

\wait{*time*}

Pauses *time* ms before continuing to print text.

\m{*name*}

\l{*name*}

Inserts the value of a name variable: \m for global names, and \l for local names. *name* can be either an integer (the index of the variable, as used in calls to the GetName() family) or one or two alphabetic characters ('A' to 'ZZ', as used in the #NAME variables in gameexe.ini). See 5.4.2 for details of name variables.

A second argument may optionally be supplied, which should be a constant integer. If this is given, it identifies a single character of the name to be printed (zero-indexed).

\i{*int*}

Displays the value of *int*, which can be an arbitrary integer expression.

You may optionally supply a minimum length, of the format \i:*length*{*int*}; if this is supplied, the number will be left-padded with zeroes to ensure that it is always at least *length* digits.

\s{*str*}

Displays the value of the string variable *str*.

\size{*[pixels]*}

pixels is optional. If it is given, the font size will be set to that size; if it is not, the font size will be reset to the default.

\c{*idx*, *[bg_idx]*}

Sets the colour of the following text. The values are indices to the game's palette, which is defined in `gameexe.ini` with the `#COLOR_TABLE` command. `\c` can also be used without any arguments, which resets the colours to their defaults.

For example, if `COLOR_TABLE.001` were red and `COLOR_TABLE.002` were green, someone addicted to pointless examples could write

```
'I like using \c{1}red\c{} text, and sometimes \
 \c{1, 2}red with a green shadow\c{}.'
```

This control code behaves in the same way as the `FontColour()` function: the colour will be reset automatically at the end of the string. If you want to set multiple strings in the same non-default colour, you will have to use `\c` at the start of all of them.

\ruby{*text*}={*gloss*}

Used to display interlinear glosses or ruby, also known as *furigana*. *text* will be displayed normally, with *gloss* printed above it in small type.

You must set the `#WINDOW.LUBY_SIZE` *(sic)* variable for the current window to an appropriate size in order to use this control code.

\e{*index*, *[size]*}

\em{...}

Prints a bitmapped character or icon at the current point in the text. `\e` prints the bitmap in full colour; `\em` takes its alpha channel and displays that in the current font colour.

Bitmaps are drawn from files defined with the `#E_MOJI` settings in `gameexe.ini`. You must define at least `E_MOJI.000` in order to use these codes. If multiple files are defined, they should contain the same characters at different sizes.

The bitmap used is always the largest available that is in height smaller than or equal to the current font size; if no matching bitmap is available, a space is left. The optional *size* argument can be used to force the font size to a particular size temporarily, to select a particular size manually. Note however that in the current implementation it also resets the font size to the window's default - you will have to follow it with a `\size` code if you were working at a different size.

\mv{*x*, *y*}

\mvx{*x*}
\mvy{*y*}

Move the insertion point (i.e. the offset of the next character) by (x, y) pixels.

If the new offset is beyond the right-hand margin of the text window, the text will automatically wrap, effectively placing it at the start of the next line instead. Other margins are not checked, but text will always be clipped to the window margins; you can place text above or to the left of the window, but only those parts of letters within the margins will be displayed.

There are three major uses for these codes. The first is character composition by overprinting:

```
#const cw = gameexe('WINDOW.000.MOJI_SIZE') / 2
        + gameexe('WINDOW.000.MOJI_REP', 0) / 2
'A t^\mvx{-cw}ete-`\mvx{-cw}a-t^\mvx{-cw}ete with Sayuri'
page
```

displays "a tête-à-tête with Sayuri" fairly reliably, even though the accented characters do not exist in most Japanese fonts. (Note the code that calculates the width in pixels of a Latin character - this would need modifying if you were not printing to window 0, or not using the default font size.)

The second use is printing superscript and subscript text:

```
'\size{26}x\size{16}\mvy{-1}2\mvy{1}\size{26}' // x^{2}
'  +\mvx{8}'                                  // +
'log\mv{1,13}\size{18}16\size{26}\mv{6,-13}y' // \log_{16} y
'\mvx{8}=\mvx{8}z\size{}'                     // = z
page
```

displays a reasonably nicely spaced version of "$x^2 + \log_{16} y = z$" in most fonts (observe the use of \mvx{8} to insert a narrower-than-usual space).

The final use is printing non-square bitmapped characters: `'\e{0}\mvx{-4}'` would be a suitable way to display a bitmap containing a character four pixels narrower than the font height.

These codes are also used by the rlBabel DLL to implement proportional text output (see 7.1.2 and the rlBabel source code for implementation details).

\pos{*x*, *y*}

\posx{*x*}
\posy{*y*}

As \mv etc., except that the coordinates are interpreted as absolute offsets from the origin, rather than relative to the current insertion point.

\ind{*[pixels]*}

pixels is optional. If it is given, indentation will be set to that value; if it is not, indentation will be reset to the default.

\noind

Reset indentation.

\shake{*index*}

Shakes the screen, using the movements defined in `gameexe.ini` with the variable @irefSHAKE.*index*. #SHAKE.*index*.

pixels is optional. If it is given, indentation will be set to that value; if it is not, indentation will be reset to the default.

4.5.4 Names

As display strings are normally used for game text, RealLive naturally provides means for identifying character names. This is accomplished with the \name control code.

The format of this code is \{*text*}. *text* is arbitrary text, which is used as the current character name.

Note two unique features of this code. Firstly, it has no identifier; you can use \name{} as well, but \{} is the canonical form. This is just to save typing, as this is the most common control code. Secondly, *if it is followed by a space, that is gobbled.* This permits you to write `'\{Foo} "..."'`, and have it appear correctly spaced in the output. If you actually wanted a space after the name, escape it with another backslash: `'\{Foo}\ "..."'`. (You probably won't like the results.)

The effect this has is can be controlled to a great extent by settings in `gameexe.ini`. If `#WINDOW.NAME_MOD` is non-zero for the current window style, the name will be displayed in a separate smaller window, based on the other `#WINDOW.NAME` settings. If `#WINDOW.NAME_MOD` is zero, the name will be printed inline at the current position, and followed with a space; in this latter case, if `#WINDOW.INDENT_USE` is non-zero, an indent point will then be set at the new *x* offset in the text window, meaning that subsequent lines of text will begin at that offset.

A separate but related topic is the use of name variables to store customisable character names. As described in section 5.4.2, these can be referenced directly from within display strings with the `\l` and `\m` control codes. For example, a common idiom to set the speaker name to the player is the code \{\m{A}}.

4.5.5 Glosses

Kepago provides for hypertext glosses using a control code \g. The syntax is \g{*text*}={*gloss*}. *text* is arbitrary text, which is always output in the normal way, and may be highlighted in some way. When it is clicked on, the value of *gloss* is passed as a string to a handler routine, which would typically display it in a subsidiary window.

Support for these is limited in Rlc. The control code is always accepted, but it is only processed when the rlBabel library is in use for dynamic lineation (see 7.1.2); in all other cases, *text* is simply displayed as normal text.

4.5.6 Lineation

Since the RealLive system is designed for use with Japanese only, it does not implement the more complex line breaking logic required for Western languages. In Japanese, it is acceptable to break a line anywhere, including in the middle of words, so this is the behaviour RealLive adopts. This makes it necessary to implement lineation specially for Western text.

Versions of RLdev up to 1.03 implemented static lineation. Since this is impossible to do correctly, however, this feature has been removed and replaced with various dynamic lineation techniques, which ensure that text is always correctly lineated even in the presence of variable-length elements. These are implemented as libraries, and disabled by default; see section 7 for usage details.

4.6 Resource files and resource strings

It can often be desirable to separate program logic from the game's script; for example, if one is releasing a game in several languages, it is more convenient to provide translators with just the text, while if one is releasing versions of the same game with different code (for example, adult and all-age versions) it can be convenient to be

able to use the same script with both. Resource files provide a simple means of accomplishing this.

4.6.1 Resource file syntax

A resource file is basically an association table of keys to strings.

A key is defined by enclosing it in angle brackets. Keys can contain any combination of characters, with a few restrictions. Firstly, purely numerical keys are treated as numbers: <$00d> identifies the same string as <13> and <0013>. Secondly, keys may not contain spaces, line breaks, or the characters > and }. Both of these restrictions can be worked round by quoting the key; such a key uses the normal string literal syntax, can contain any character, and distinguishes between different representations of integers.

Each occurrence of a key begins a new string; the text between it and the next key or the end of the file is interpreted as a resource string. The syntax of a resource string is broadly similar to that of a display string literal (see section 4.5), with a few exceptions:

- The literal character < must be written \<.
- Kepago comments are parsed, so literal // must be written \// and literal {– must be written {\–. For block comments, note that control codes take precedence: that is to say, \{– will be interpreted as the opening of a name block (see 4.5.4) that begins with a – character, and \i{–1–}{1} will raise a syntax error, since it will be interpreted as \i{ –1 – } followed by the text {1}, and Kepago requires a right-hand side to what is being read as a subtraction operator.
- Quotes are always treated as literal characters, and never need escaping.
- Line breaks do not need escaping; however, any whitespace before an unescaped line break will be trimmed,[1] so you can escape the line break to preserve it. Alternatively, _ can be used to represent a non-trimmable space.

4.6.2 Additional control codes

There are also some additional control codes that can be used in resource strings.

\d

Where two versions of a script are being produced, and one requires fewer strings than the base version, you can use `\d` to remove superfluous strings without modifying the Kepago source code.

The resource string becomes a 'remove string' command; referencing it will remove the reference, and if it is referenced by a display string command that is followed by a `pause()` call, the pause will also be removed.

\a{*[str]*}

Where two versions of a script are being produced, and one requires more strings than the base version, you can use `\a` to add extra strings without modifying the Kepago source code.

[1] LaTeX users may expect spacing to be automatic, but the syntactic similarities to LaTeX are coincidental; 'and!the', where the ! is a line break, will produce 'andthe', not 'and the'.

The current resource string is processed as normal, but when it is referenced by a display string command, the resource string <*str*> will be added after it as another display string command; if the referencing command was followed by a `pause()` call, another `pause()` will be added after the new string. If multiple `\a` codes appear in one resource string, the extra strings will be added in the order of the `\a` codes.

str is optional. If it is not given (an 'anonymous reference'), the next string in the resource file will be used; see 4.6.3 below for how multiple anonymous references are resolved.

\res \res{*ID*}

Where multiple versions of a script are being produced, where a variant contains many strings in common with a base version and the resource ids vary from base to variant, it may simplify a things considerably to inherit common resource strings from the base.

You can use `\res` to specify the id of a resource string to inherit from a base resource file loaded using a `#base_res` directive in the Kepago script file.

If no `#base_res` directive was used, ids refer to strings within the current resource file.

4.6.3 Anonymous references

If multiple anonymous references are given in a single string, they are resolved sequentially and recursively: that is, anonymous references within a string referenced anonymously are resolved *before* any further anonymous references in the original string.

An example may make this clearer:

```
<foo>
  This is foo.\a\a
<bar>
  This is bar.\a
<baz>
  This is baz.
<quux>
  This is quux.
```

This is equivalent to writing

```
<foo>
  This is foo.\a{bar}\a{quux}
<bar>
  This is bar.\a{baz}
<baz>
  This is baz.
<quux>
  This is quux.
```

Note how the second `\a` in <foo> resolves to <quux>, because <baz> has been taken by the `\a` in <bar>.

Resource strings for use with anonymous references can be anonymous themselves: that is, you could also write

```
<foo>
  This is foo.\a
<>
  This is anonymous.
```

This can be clearer to read, and has the advantage that anonymous resource strings normally generate an error, so you will be able to tell whether you have included the correct number.

4.6.4 Using resource strings

To load a resource file and make the strings it contains available to your code, use the `#resource` directive at the start of the Kepago source file.

To reference a resource string, use the `#res` directive with the key of the string you wish to include.

To load a base resource file and make the strings it contains available to your code, use the `#resource` directive at the start of the Kepago source file.

To reference a base resource string within a resource file, use a `\res` control code with the key of the base resource string you wish to include.

Chapter 5

The RealLive system

5.1 Overview

RealLive is an engine designed to power *bishōjo* games such as visual novels and simple simulations; it is based around a fast Turing-complete bytecode interpreter, and provides functionality for text and graphics, sound, music, and simple animation, along the lines that such games require. Notable products based on the RealLive engine include Key's *Clannad*, *Planetarian*, and *Kanon Standard Edition*, and 130cm's *Princess Bride* (not to be confused with the book/film of the same name).

A RealLive game is made up of four main parts: the interpreter (`avg2000.exe`, `reallive.exe`, or `kinetic.exe`), the configuration file (`gameexe.ini`), the scenario file (typically `seen.txt`[1]), and accompanying data files, which vary in type and number, but typically include graphics (in the `g00` and `pdt` formats), animations (in the `gan` and `anm` formats), and music (in the `nwa` format). Of these parts, RLdev deals only with the scenario file and graphics, although Rlc reads certain settings from `gameexe.ini` when compiling.[2]

Note that in the case of DRM-protected games, the configuration and data files are further compacted into a single file (typically `kineticdata.pak`), which is encrypted using a rather stronger method that I have not yet discovered how to unlock. It is not possible to access the files this contains directly with RLdev, although it is simple enough to extract the contents manually from a memory dump of a running game.

There exists a clone of the RealLive interpreter, Jagarl's `xclannad` (currently at `http://www.creator.club.ne.jp/~jagarl/xclannad.html`). The latest version at the time of writing, 0.06, did not implement enough features to be a viable replacement, but it promises to become such with time.

5.1.1 Targets and versions

The family of interpreters I refer to here as RealLive actually comprises three more-or-less distinct interpreters: AVG2000, RealLive, and Kinetic. While the API and basic

[1] While other names are possible, I don't know of any games that use them. The name 'seen' appears to derive from a misunderstanding — the word intended is clearly 'scene', and in Japanese the two are of course not only homophones but homographs as well.

[2] Utilities to handle the remaining proprietary formats are planned, but at present those wishing to modify resources other than bytecode and bitmaps must look elsewhere.

bytecode format used by all three is essentially identical, they are not compatible: AVG2000 (the original successor to AVG32, later renamed to RealLive) uses a different scenario header and encoding scheme, and Kinetic (a special DRM-enabled interpreter used only for the Kinetic Novel series) reassigns a large number of fundamental operations in what appears to be a futile attempt to make reverse-engineering harder.

To complicate matters, however, the API itself is under constant development. There are significant differences between versions. For example, the `grpRotate()` functions were introduced in 1.1.5, and auto mode early in the 1.2 series. There is no fixed mapping between the API/bytecode version (the "version") and the header/basic operation types (the "target"): the AVG2000 interpreter was retired at around version 1.0.0.8, and the Kinetic interpreter introduced at version 1.2.6.4, but RealLive-proper interpreters can be found at all stages of development.

When compiling for a RealLive interpreter, therefore, the target and version must both be chosen separately. Bytecode compiled for a 1.1-series RealLive will normally run perfectly on a 1.3-series RealLive,[3] but the converse is not true, and bytecode compiled for RealLive 1.2.7.0 is incredibly unlikely to run in Kinetic 1.2.7.0. Since different versions of games sometimes use incompatible interpreters, you will probably have to recompile code with different flags or directives if you are working on such a game. In most cases, however, Rlc will manage to generate suitable code for any interpreter from any source code, given the correct compilation settings.

Except where otherwise specified, the version documented here is basically 1.2.6.8 (as supplied with *Kanon Standard Edition*). Other versions have not been exhaustively tested: differences and limitations are documented where I'm aware of them, but in general you should not assume that any feature mentioned in this manual is necessarily available, particularly if you're working with a 1.0- or 1.1-series interpreter. Note that very few tests have been carried out with the 1.0.0.8 interpreter, and practically none with AVG2000.

5.2 Scenarios

The scenario file contains the game logic. It is a simple archive of up to 9,999 individual compilation units, termed 'scenarios', which are named `seen0001.txt` to `seen9999.txt`; each of these is an individually compressed and self-contained block of RealLive bytecode.

The scenarios represent the major divisions of a program; only code from one scenario can be accessed at any one time, though switching between them can easily be done with the `jump()` and `farcall()` functions, and up to 100 entrypoints may be defined within each scenario, effectively permitting access to arbitrary locations. A common idiom is to define several related functions (particularly where they share code) in one scenario, and use `farcall()` with an entrypoint index to access them from the rest of the game; this works on the same principle as linking a library into a C program, but in Rlc it must be done by hand.

It is not necessary to build a scenario file in order to run code: if a file of the form `seenNNNN.txt` exists in the same directory as the scenario file, its contents will override the scenario of that number. This can be convenient when debugging, and is the only trivial way to inject custom code into a DRM-protected game.

[3] There have been several cases of API breakage, though; the DLL interface was changed incompatibly around version 1.2.5, and the `grpDisplay()` function around 1.0.9, to name but two.

5.3 Debugging

The RealLive interpreter contains a convenient debugger, which can be enabled by defining `#MEMORY` in `gameexe.ini`. The various debugging features can be accessed with function keys or from drop-down menus. Of particular use are the single-step execution mode (`F3`), the memory editor (`F5`) and the graphics DC viewer (`F7`).

Debug mode also enables a number of runtime warning messages relating to matters such as memory management; these do not always represent problems, but it's probably wise to try to eliminate them anyway.

Finally, there are various message-box and input-related functions which are only processed in debug mode. See 6.15 for a full list.

From version 1.3 onwards, single-step mode is replaced with a simple source debugger. This is usable with Kepago - you must rename your source file to have the extension `.ORG` and set up the path to that file in the debugging options dialog within the interpreter - but does not work very well with Kepago features like header files, and it is *not* likely to be usable with disassembled code (in which lineation information is never preserved), so its utility to developers using RLdev is limited. It does, however, provide a slightly more intrusive form of single-step execution regardless of whether source code is available or not.

5.4 Memory

Only statically allocated memory is available, in the form of a number of arrays of variables; there is no stack.

Variables are divided into 'local' and 'global' types. The values of local variables are reset when the program is executed, and stored in saved games. Global variables are persistent, and their values are shared between all saved games.

5.4.1 Integers

Integers

RealLive provides unsigned 1, 2, 4, and 8-bit integers, and signed 32-bit integers; these share the same memory, so individual bits and bytes can be examined and modified without resorting to bitwise operators and shifts. There are eight separate integer spaces, each of 8,000 bytes.

Local 32-bit integers are stored in A[0] to A[1999], and likewise B[] to F[]; G[] and Z[] are global equivalents.

The smaller elements are accessed through arrays A8b[], A4b[], A2b[], Ab[], and similarly named equivalents for B[] to G[] and Z[]. Indexing is based on the element size: the 8-bit arrays have elements from 0 to 7,999, arranged such that the four bytes of A[100] can be accessed as A8b[400] to A8b[403], in the little-endian order; this pattern is consistent, so for example Gb[42784] shares the same memory as the least significant bit of G[1337].

These memory spaces are accessed through variables, either those that you define yourself, or a set of built-in arrays that correspond directly to the memory spaces. To avoid restricting useful single-character identifiers, Rlc adds the prefix 'int' to the names in declaring these arrays. For example, the memory cell A[100] may be accessed by referencing the variable intA[100].

The store register

There also exists a special integer variable store. This is a register used to return values from functions such as `strlen()` and `select()`; in all other respects it behaves exactly like any other variable.

While technically these functions do not have return values, Kepago permits you to treat them as though they did: for example, the code that RealLive bytecode represents literally as

```
strlen(strS[100])
intA[10] = store
```

can also be written

```
int x -> MEMARR_A.10
str s -> MEMARR_S.100
x = strlen(s)
```

It is sometimes more efficient to access store directly:

```
strlen(s)
if store > 5 && store < 10...
```

generates code marginally more efficient than either repeating the `strlen()` call or assigning its value to a variable would. Note however that Rlc makes no guarantee that it will not generate code that affects the value of store; in general you cannot assume that its value will remain unchanged between two statements.

5.4.2 Strings

String variables

RealLive strings are null-terminated character arrays; allocation is handled automatically. Their length is not limited at runtime, but only up to 10,000 characters are stored in saved games.

The array S[0] to S[1999] holds local string variables. In RealLive (but not in AVG2000) there is also an array M[], of the same size, the contents of which are global.

As with integers, this memory can be accessed either through variables you declare or through two built-in arrays, strS[] and strM[].

Name variables

In addition to the normal string variables, there also exist some special variables designed primarily to hold character names. These cannot be accessed directly in source code, but they can be included inline in strings.

There are 702 global name variables; they can each hold between 12 and 20 characters, depending on the `#NAME_MAXLEN` setting. Their default values are set in `gameexe.ini` with the `#NAME` variables (`NAME.A`, `NAME.B`, and so on). Their values can be read and modified with the `GetName()` and `SetName()` functions. Within strings, they are included with the control code `\m`: the first is \m{A}, the second \m{B}, through \m{Z}, \m{AA} to \m{AZ}, and so on up to \m{ZZ}.

Names can also be accessed numerically, such that 0 is A, 26 is AA, and 701 is ZZ. The numerical form is valid everywhere but in `gameexe.ini`, and the letter form is valid everywhere but in function parameters.[4]

In RealLive (but not in AVG2000) there is a parallel set of local name variables, which are to the normal set as S[] is to M[]. These are introduced inline with `\l` in place of `\m`, the getter/setter functions are `GetLocalName()` and `SetLocalName()`, and the `gameexe.ini` variables defining their default values are called `#LOCALNAME`.

For example, *Clannad* uses the global names A and B for the player's family name and personal name respectively, and various local names for certain addresser/addressee combinations that have to change over the course of the game, such as the way the player addresses Nagisa (\l{C}).

5.4.3 Call variables

Version 1.3 of RealLive introduced a set of "call variables": these are K[] (three string variables) and L[] (40 integer variables). These can be used as ordinary variables, but they are intended for use with the functions `gosub_with()` and `farcall_with()`.

They can be accessed from Kepago with two more built-in arrays, strK[0] to strK[2] and intL[0] to intL[39].

5.5 The system command menu

RealLive provides a context menu system to handle most actions and configuration settings. The system command menu is configured with the `#SYSCOM` variables in `gameexe.ini`. It can be disabled by setting `#SYSCOM_USE` to 0, and if a `#CANCELCALL` hook is defined it will never be used at all (*Clannad* does this, although it uses the internal flags associated with the system command menu to control its own menu system).

The menu is displayed manually by right-clicking or pressing Escape. It can be displayed from code with the `ContextMenu()` function.

The shape of the menu is determined by the `gameexe.ini` variables `#SYSCOM_MOD`, `#SYSCOM_MOD2`, and `#SAVELOADDLG_USE`.

The system commands, and their equivalents in code, are listed below. They can be managed with the functions described in 6.14.6. Items marked with an asterisk have standard dialog boxes which can be accessed with `InvokeSyscom()`; items marked with a plus sign have settings which can be modified with `InvokeSyscom()` and retrieved with `ReadSyscom()` (this category may be incomplete). Functions related to settings come in sets including a getter, a setter, and sometimes a function to get the default value. For example, the getter function `MessageSpeed()` is accompanied by `SetMessageSpeed()` and `DefMessageSpeed()`. Refer to the individual functions in the API reference for full details.

[4] You will almost invariably find yourself having to use both forms at some point in a project. It may help to think of the letter form as using letters for numbers in a base-26 encoding. Or then again it may not.

0 Save
menu_save()

1 Load
menu_load()

2 * Message speed
`MessageSpeed()`

3 * Window attributes
`GetWindowAttr()`; also `WindowAttrR()`, etc.

4 * Volume settings
`BgmVolMod()`, `PcmVolMod()`, `KoeVolMod()`, and `SeVolMod()`; likewise
`BgmEnabled()`, etc.

5 + Display mode (full-screen or windowed)
`ScreenMode()`

6 * Miscellaneous settings
`CursorMono()`, `SkipAnimations()`, `LowPriority()`, `ConfirmSaveLoad()`, `ReduceDistortion()`, and `SoundQuality()`

8 Voice settings (whether to use text, voice, or both)
`KoeMode()`

9 * Font selection

10 * BGM fade (whether to fade out when voice is playing)
`BgmKoeFade()`, `BgmKoeFadeVol()`

11 BGM settings (DirectSound or CDDA)

12 Window decoration style
`GetWakuAll()`

13 * Auto mode settings
`AutoCharTime()`, `AutoBaseTime()`

14 Return to previous selection point
`ReturnPrevSelect()`

15 Enable/disable character voices
`UseKoe()`

16 * Display game version

17 Enable/disable environmental effects
`ShowWeather()`

18 Show/hide object 1
Meaning is application-defined; in *Clannad*, this is the date window setting.
`ShowObject1()`

19 Show/hide object 2
Meaning is application-defined.
`ShowObject2()`

20 Enable/disable colour-based text classification
Meaning is not fully understood, but it may be something to do with displaying text in a different colour if it has already been viewed, or using different colours for different characters.
`ClassifyText()`

21 Generic setting 1
Meaning is application-defined.
`Generic1()`

22 Generic setting 2
Meaning is application-defined.
`Generic2()`

24 Open file
Opens the file named by `#MANUAL_PATH` in `gameexe.ini`.
This function is not available in all RealLive builds; it was introduced between versions 1.2.3 and 1.2.6. It is not clear whether it can be accessed other than through the right-click menu.

25 Skip read text
`SetSkipMode()`

26 Enable auto mode
`AutoMode()`

28 Return to main menu
`MenuReturn()`

29 Exit game
`end()`

30 Hide menu

31 Hide text window
`ShowBackground()`

5.6 Extension DLLs

Version 1.2 of RealLive introduced a plugin system to permit arbitrary code to be called from external DLLs. (This feature does not exist in AVG2000 or earlier versions of RealLive.)

To write an extension DLL, simply write regular C++ code that conforms to the interface documented here, and load it into the game as described below.

5.6.1 Using an extension DLL

The method for telling RealLive to use functions from a DLL depends on the version of the interpreter in use.

For versions between 1.2 and 1.2.5, you must load it at runtime with the `LoadDLL()` function. These versions only support the use of one DLL at a time.

From version 1.2.5 onwards, the preferred method is to use a `#DLL` variable in `gameexe.ini`; such DLLs will then be loaded automatically and kept in memory throughout execution, which is generally more useful. The `LoadDLL()` interface is deprecated in these versions, and indeed is removed altogether from version 1.3.2 onwards.

It is possible for a DLL to be useful simply by being in memory; an example of this is the rlBabel library (7.1) supplied as part of RLdev. However, it is more usual for it to be used to provide an extended API to RealLive bytecode. This is accessed using the standard function `CallDLL()`.

5.6.2 The extension DLL interface

There are two distinct interfaces used by different versions of RealLive. RLdev provides an abstraction which will enable the same source code to be compiled against both interfaces, within limits.

To write an extension DLL, include the header `rlplugin.h` found in the `lib/` directory of your RLdev installation. By default, the new interface is used; to use the old interface, define the symbol OLD_INTERFACE when compiling your code.

Follow the instructions provided with your compiler to produce a DLL. If you need a definition file, two are provided: `rlplugin.def` (for the new interface) and `rlpluginf.def` (for the old interface).

OnLoad function

The OnLoad function is called once when the DLL is loaded into memory.

long OnLoad(RealLiveState *State, size_t cbSize);

Pointer to a RealLiveState structure (see below, 5.6.2) containing the internal state of the interpreter. The values this contains remain valid until the DLL is unloaded.

The size of the structure, in bytes.

*State

Return value

The function should return a non-zero value to indicate that the DLL was loaded successfully.

OnFree function

The OnFree function is called once when the DLL is unloaded.

long OnFree();

Return value

The function should return a non-zero value to indicate that the DLL was unloaded successfully.

In RealLive 1.2.5 and up, the unload event happens only when the DLL is explicitly unloaded with `UnloadDLL()`, or when the interpreter closes.

In older versions, however, the DLL is unloaded whenever the interpreter is reset. Unfortunately, actions that trigger a reset include the loading of a saved game, and any DLL that had been present when the game was saved is *not* reloaded. This can be problematic if your code relies on the DLL being persistent. The simplest workaround is to ensure that any call to the DLL returns only non-zero values: a zero return value is then a certain indication that the DLL needs reloading.

OnInit function

In RealLive 1.2.5 and up, the OnInit function is called each time the interpreter is reset (see immediately above).

This function does not exist in older versions of RealLive.

void OnInit();

Return value

This function does not return a value.

OnCall function

The OnCall function is the entrypoint for calls from RealLive bytecode (see `CallDLL()`).

long OnCall(**long** arg1, **long** arg2, **long** arg3, **long** arg4, **long** arg5);

The arguments passed to the `CallDLL()` function. Any arguments that were not provided will default to 0.

The meaning of these arguments is entirely up to you. Typically the first argument will be used as a function identifier, and the OnCall function will dispatch control to other functions in your DLL accordingly.

Return value

The value returned by this function is entirely up to you. It will be used as the return value to the `CallDLL()` call.

You may wish to reserve the value 0 as an error code to indicate that there is no DLL loaded.

RealLiveState structure

The RealLiveState structure is the official interface by which extension DLLs can access the internal state of the interpreter.

```
struct RealLiveState {
  size_t  cbSize;
  HWND hMainWindow;
  long *intA;
  long *intB;
  long *intC;
  long *intD;
  long *intE;
  long *intF;
  long *intG;
  long *intZ;
  char** *strS;
  char** *strM;
  struct {
    void** *pData;
    long *pWidth;
    long *pHeight;
  } BankInfo[16];
  char* szGamePath;
  char* szSavePath;
  char* szBgmPath;
  char* szKoePath;
  char* szMovPath;
  char* szDataPath;
};
```

The size of the structure, in bytes.

Handle to the main game window.

The integer memory space. Accessing intA[1000] has exactly the same meaning in an extension DLL as it does in Kepago.

The string memory space. Each element is a pointer to a null-terminated string. It is unclear whether it is safe to make any assumptions about the size of the buffer pointed to, or whether it is safe to reallocate the memory.

An array of internal structures that provide direct access to the game's graphics DCs. *pData is a pointer to the raw pixel data, and the other two members are pointers to the dimensions of that data, probably in pixels.

The game root directory (i.e. the path of `gameexe.ini`).

The save directory.

The directories where music, voice, and movie data are stored (that is, the root for the `FOLDNAME` paths for them).

The base directory where all other resources are sought (that is, the root for the `FOLDNAME` paths for them).

Chapter 6

The Kepago/RealLive API

6.1 Introduction

First, an important note: parts of the RealLive interpreter are documented within this chapter. If looking up individual functions, you should also make sure you read the introductions to each section and subsection, where general issues relating to functions of each type are explained. This is particularly important for the sections on graphics (6.10) and objects (6.12), since in those sections many functions whose names and properties can be derived trivially from other functions are not explicitly documented (for example, the function `recCopy()` is documented, but the function `grpCopy()` is not documented explicitly because its existence and exact behaviour are both made clear by the existence of `recCopy()`).

While most of the functions documented here are internal RealLive functions exposed by the Kepago/RealLive API, you should bear in mind that this API differs in some places from the underlying system. In addition to having a number of additional functions implemented in Kepago, it also introduces a number of abstractions, such as representing multiple definitions with optional parameters and single result parameters as return values, which mean that the function definitions here do not have a 1:1 correspondance with the bytecode they represent. If you're doing bytecode-level hacking, or compiler or interpreter development, you should use this document as a reference only in conjunction with `reallive.kfn`, `system.kh`, `rlapi.kh`, and the Rlc source code.

The accuracy of the information in this chapter is not guaranteed. The API is based on reverse-engineering; my information is therefore limited to what I and others have been able to deduce from the bytecode of official RealLive games and by experimentation. Since no official documentation has been used, it is likely that I have misinterpreted the effects of many functions, and I know for certain that there are over a hundred more that I have not exposed or documented at all.

Where differences in behaviour or availability relating to the version of the interpreter in use are known, they are documented here. Since most of my testing has been done with RealLive, the AVG2000 data here are seriously incomplete, and I have not even attempted to note differences in `xclannad`'s behaviour, given the unfinished state of that program at the time of writing.

The definition format should be self-evident. Return value types are given after the function; 'store' denotes a function that modifies store (and can therefore be

used with or without an assignment to an integer variable). Where multiple related functions have identical parameters and return types, all but the first in a group are given '...' in place of the parameter list.

6.2 Duplicates

A number of functions, such as `load()`, have what seem to be duplicates with different opcodes but apparently identical behaviour. These duplicates have been given names with integer suffixes, such as load2, and are not otherwise documented here. If you encounter one, refer to the base function, and if you wish to know which functions I have designated thus, refer to `reallive.kfn`. I would be particularly interested if anyone were able to tell me about any differences in behaviour between functions I have grouped in this way.

6.3 Compatibility

From time to time, changes are made to the API which break backwards-compatibility with existing code.

Functions are initially deprecated; they are removed altogether only after a subsequent incompatible change. Functions that have been removed from the API can be found in the compatibility header `compat.kh`. By default, loading this header does absolutely nothing: you must define a symbol to indicate how old an API you need compatibility for:

```
#define PRE_121 // reverse changes made in version 1.21
#define PRE_120 // reverse changes made in version 1.20
#define PRE_103 // reverse changes made in version 1.03
```

This is a crutch. You should update your code to the current API rather than relying on `compat.kh`.

You can also gain access to compatibility wrappers for a few functions from the old Kepago/AVG32 API by defining KPACAPI before loading this header. This may be useful if you are writing code that will be shared between RealLive and AVG32 projects.

6.4 Compiler functions

6.4.1 Initialisation

rlcInit ()

dwSize
hMainWindow
***intA .. *intZ**
***strS, *strM**
BankInfo
szGamePath
szSavePath
szBgmPath, szKoePath, szMovPath
szDataPath

Call this function at the start of your program to initialise the Kepago/RealLive runtime library. It should be called once, and only once, when the interpreter is first launched.

Many programs will not require any initialisation, but in these cases no code is generated for this function, and programs which do require initialisation will function unpredictably without it. In particular, programs using the Textout or rlBabel libraries (see 7.1) *must* be initialised using this function.

rlcSetAllocation(*iarray*, *ifirst*, *ilast*, *sfirst*, *slast*)

Sets memory allocation parameters (see 3.0.2).

iarray selects a memory space to use for integer allocation. It should have one of the following values:

MEMARR_A	Use the range A[*ifirst*]...A[*ilast*]
MEMARR_B	Use the range B[*ifirst*]...B[*ilast*]
MEMARR_C	Use the range C[*ifirst*]...C[*ilast*]
MEMARR_D	Use the range D[*ifirst*]...D[*ilast*]
MEMARR_E	Use the range E[*ifirst*]...E[*ilast*]
MEMARR_F	Use the range F[*ifirst*]...F[*ilast*]

ifirst, *ilast*, *sfirst*, and *slast* should be constant integers in the range 0...1999. The first two have the meaning indicated in the table above, and the last two define the range in the S[] array used to allocate string variables.

This function should either be included in a project header file or called at the top of every scenario in any project that requires variables in the default ranges (C[0]...C[1999], S[1900]...S[1999]) not be treated as unused.

As an example, the defaults can be restored with the call

rlcSetAllocation(MEMARR_C, 0, 1999, 1900, 1999)

6.4.2 Symbolic manipulation

The macros and intrinsic functions described in this subsection are expanded at compile-time; they can be used in conditional compilation, or to track compiler state. I don't know if they'll be particularly useful outside the specialised cases they were implemented for.

defined?(*symbols*...): integer constant

Expands to 1 if all the symbols given are present in the symbol table when the macro is parsed, or 0 if any of them is not.

#ifdef Foo is literally a shorthand for *#if* defined?(Foo).

default(*symbol*, *expr*): expression

If *symbol* is defined, expands to *symbol*, otherwise expands to *expr*. *expr* does not have to be a constant expression.

constant?(*exprs*...): integer constant

Takes a list of expressions, and expands to 1 if they are all constant expressions (see 4.3), or 0 otherwise.

integer?(*exprs*...): integer constant

Tests whether parameter list consists entirely of integer expressions. Result is 1 if true, or 0 otherwise.

array? (*symbols*...): integer constant

Intrinsic macro: expands to 1 if all *symbols* are arrays, 0 otherwise.

length (*array*): integer

Returns the length of *array*, which must be an array variable.

__deref(*integer*, *expr*): integer element

[description]

__sdref(*integer*, *expr*): string element

[description]

__variable?(*expr*): integer constant

[description]

__addr(*variable*): result

[description]

__ident(*string*): result

[description]

at(*location*, *expr*): result

[description]

__empty_string?(*string*): integer constant

The present Kepago implementation does not permit comparison of strings, even constant strings, in constant expressions. This macro can be used instead to determine at compile-time whether a string constant is equal to the empty string or not, which is the most useful case.

__equal_strings?(*string*, *string*): result

[description]

gameexe(*key*, *[index]*, *[default]*): expression

Expands (at compile-time) to the value bound to *key* in `gameexe.ini`.

key should be a string constant. You can include or omit the initial # character as you wish, and keys are not case-sensitive.

For keys with multiple values (such as `#CANCELCALL`), the value returned is specified by *index*, which defaults to 0 (the first value).

If *default* is given, it is returned when *key* is not found in `gameexe.ini`. It has no default value: if *key* is not found when *default* is not given, an error is reported instead.

target_lt(*version*): result

[description]

target_le(*version*): result

[description]

target_gt(*version*): result

[description]

target_ge(*version*): result

[description]

kinetic?(): result

[description]

rlc_parse_string(*string*): statements or expression

Passes the constant string *string* to the Rlc parser. If the call is within an expression, *string* must represent a complete expression; if the call is as a statement, *string* must represent one or more complete statements. The parsed code will replace the call in the AST.

The use of this function can be considered 'deep magic'. If you find yourself considering it, think very hard about whether you really want to do whatever it is you're attempting.

const_eq?(*expr*, *const*): integer constant

Expands to 1 if *expr* is constant and equal to *const*.

This is useful when compiling conditionally:

```
#if foo == 1 // ERROR if foo is not constant
  // do stuff
#endif
#if const_eq?(foo, 1) // Never an error
  // do same stuff but safely
#endif
```

__empty_string?(*string*): integer constant

The present Kepago implementation does not permit comparison of strings, even constant strings, in constant expressions. This macro can be used instead to de-

termine at compile-time whether a string constant is equal to the empty string or not, which is the most useful case.

rlc_parse_string(*string*): statements or expression

Passes the constant string *string* to the Rlc parser. If the call is within an expression, *string* must represent a complete expression; if the call is as a statement, *string* must represent one or more complete statements. The parsed code will replace the call in the AST.

The use of this function can be considered 'deep magic'. If you find yourself considering it, think very hard about whether you really want to do whatever it is you're attempting.

6.4.3 Compile-time `gameexe.ini` access

It can be useful to include data from `gameexe.ini` in your programs, and, in the absence of any way to read it at runtime, compile-time access is usually acceptable. If you rely on these functions, ensure that the `gameexe.ini` Rlc uses for the data is the same as the one you ship.

in_gameexe?(*key*): integer constant

Expands to 1 if the variable *key* is defined in `gameexe.ini`, or 0 if it is not.

gameexe(*key*, *[index]*, *[default]*): expression

Expands (at compile-time) to the value bound to *key* in `gameexe.ini`.

key should be a string constant. You can include or omit the initial # character as you wish, and keys are not case-sensitive.

For keys with multiple values (such as `#CANCELCALL`), the value returned is specified by *index*, which defaults to 0 (the first value).

If *default* is given, it is returned when *key* is not found in `gameexe.ini`. It has no default value: if *key* is not found when *default* is not given, an error is reported instead.

6.5 Flow control

6.5.1 Termination

end()

Exits the interpreter.

halt()

Abnormal termination. RealLive halts without exiting, to permit debugging.

6.5.2 Inter-scenario jumps

jump(*scenario*, *[entrypoint]*)

Jumps to *entrypoint* in *scenario*. If *entrypoint* is not given, it defaults to 0.

Entry points are defined with the #entrypoint directive.

farcall(*scenario*, *[entrypoint]*)

Pushes the current location onto the call stack, then jumps to to *entrypoint* in *scenario*. Call rtl() to return from the call.

rtl()

Returns from the most recent farcall() call.

Note that gosub/ret() and farcall()/rtl() calls share the same call stack, so they must be paired properly. An error occurs if you call rtl() when the top of the call stack is a gosub call.

farcall_with(*scenario*, *entrypoint*, *parameter*...): store

This function is not available in RealLive prior to 1.3.

As farcall(), but passes parameters to the called scenario. Each *parameter* is placed in the next available call variable (see 5.4.3) of the appropriate type. It probably follows that up to 40 integer and 3 string parameters can be passed.

farcall_with(*[value]*)

As rtl(), with an extension: if *value* is supplied, and the caller was farcall_with(), then *value* is placed in store and used as the return value of the farcall_with() call.

ReturnMenu()

Returns to the main menu. This function is equivalent to jump(menu), where *menu* is defined by #SEEN_MENU in gameexe.ini.

MenuReturn()

Similar to ReturnMenu(), but the screen fades out first. This is the function called when you select syscom 28 “Return to menu” from the system command menu.

rtlButton()
rtlCancel()
rtlSystem()

These appear to behave identically to rtl().

rtlButton() is used in *Princess Bride* to return from the handling routines for extra window buttons (as defined with #WBCALL in gameexe.ini), and rtlCancel() is used in *Clannad* to return from the in-game menu (as defined with #CANCELCALL).

6.5.3 Local jumps

goto @label

Jumps to the label @label in the current scenario. See section 4.4.2 for details on labels.

gosub @label

Pushes the current location onto the call stack, then jumps to the label @label in the current scenario. Call `ret()` to resume execution at the next statement.

ret()

Returns from the most recent `gosub` call.

Note that `gosub`/`ret()` and `farcall()`/`rtl()` calls share the same call stack, so they must be paired properly. An error occurs if you call `ret()` when the top of the call stack is a `farcall()` call.

gosub_with (*parameter*...) @label: store

This function is not available in RealLive prior to 1.3.

As `gosub()`, but passes parameters to the called subroutine. Each *parameter* is placed in the next available call variable (see 5.4.3) of the appropriate type. It probably follows that up to 40 integer and 3 string parameters can be passed.

ret_with(*[value]*)

As `ret()`, but with an extension: if *value* is supplied, and the caller was `gosub_with()`, then *value* is placed in store and used as the return value of the `gosub_with()` call.

goto_if (*condition*) @label
goto_unless (*condition*) @label

Conditional equivalents of `goto`: `goto_if()` jumps to @label if the value of *condition* is non-zero, and `goto_unless()` jumps if it is zero.

`goto_if()` is not available in all interpreter versions.

gosub_if (*condition*) @label
gosub_unless (*condition*) @label

These are to `gosub` as `goto_if()` and `goto_unless()` are to `goto`.

goto_on (*expr*) { @label0, @label1, ..., @labeln }
gosub_on...

Table jumps. *expr* is evaluated, and control passed to the corresponding label in the list, counting from 0. If *expr* falls outside the valid range, no jump takes place, and execution continues from the next statement instead.

`goto_on()` uses `goto` to jump, and `gosub_on()` uses `gosub`.

The maximum table size is unknown; I have successfully generated tables with 5,000 entries. Lookups should be O(1) for any sane implementation.

goto_case (*expr*) { *val1*: @label1; *val2*: @label2; ...; _: @default }
gosub_case...

Conditional table jumps. *expr* is evaluated, and compared to *val1*, *val2*, etc. in turn, and control passes to the label associated with the first matching value.

The default case is indicated with _. This must always be present, and it must always be last. It is used if no other value matches.

`goto_case()` uses `goto` to jump, and `gosub_case()` uses `gosub`.

As with `goto_on()`, the maximum number of cases is unknown, but definitely exceeds 5,000. It is likely that these functions do not scale so well, however. If the implementation is naive, lookups will be O(n).

6.5.4 Call stack and execution state

SceneNum(): store

Returns the index of the current scenario.

CallStackSize(): store

Returns the current length of the call stack.

CallStackPop(*[count]*)

Removes the top *count* entries from the call stack (the default is 1).

CallStackTrunc(*length*)

Truncates the call stack by discarding all but the last *length* entries. That is, CallStackTrunc(len) is equivalent to CallStackPop(CallStackSize − len).

CallStackClear()

Removes all entries from the call stack.

CallStackNop(*[count]*)

Adds *count* dummy calls to the top of the call stack. (It's not clear why you would ever want to.)

6.5.5 Interrupts

Version 1.3.1 introduces a mechanism for calling a subroutine repeatedly in the background. The functions in this section are not available in versions prior to this. (The mechanism seems to have been introduced in 1.2.9, but I have not yet got it working in any of the 1.2.9 interpreters I have.)

Such subroutines should be defined by their own entrypoints. There are certain limitations in the functions that can be used in them. Translated roughly from the information screen relating to them:

* Objects and memory can be modified, and jumps and calls are permitted. However, the screen cannot be rendered to directly. (It appears that offscreen DCs can be written to, and possibly DC 0 provided that no redraw is required?)

* The keyboard, mouse, and timer status may be queried, but functions which pause first (like `GetClick()`) or modify the stored status (like `FlushClick()`) are not supported. (This restriction seems to have been lifted as of 1.3.2)

* The effect of the DISP command [possibly `refresh()`, this needs testing] is changed. Since interrupt procedures are actually triggered by DISP calls in the first place, a DISP call has the same effect as `yield()`, i.e. it returns from the interrupt.

* Various other commands are also permitted, basically those that do not cause the interpreter to break out of the event loop. Pauses, wipes, text display, and selection commands are right out.

Note that as of version 1.3.2 this mechanism was still marked as experimental; you should probably treat it with caution. Note also that I have not tested these functions much, so this documentation is probably even less reliable.

SetInterrupt(*scenario*, *entrypoint*)

Sets an interrupt on the given entrypoint. If I understand the above correctly, this will be called every frame, or every time `refresh()` is called.

ClearInterrupt()

Clears a previously set interrupt.

yield()

Returns from an interrupt handler. Call this instead of `rtl()`.

6.6 Variable manipulation

6.6.1 Integers

rnd(*[min]*, *max*): store

Returns a random integer between *min* and *max* inclusive; *min* defaults to 0 if not given.

min(*a*, *b*): store

Returns the lesser of *a* and *b*. This function can be included in constant expressions if both parameters are constant.

max(*a*, *b*): store

Returns the greater of *a* and *b*. This function can be included in constant expressions if both parameters are constant.

constrain(*min*, *x*, *max*): store

If $min \leqslant x \leqslant max$, returns *x*; otherwise returns whichever of *min* or *max* is closer to *x*. This function can be included in constant expressions if all three parameters are constant.

pcnt(*numerator*, *denominator*): store

Returns the percentage of *denominator* represented by *numerator*, i.e. $\frac{numerator}{denominator} \times 100$

sign(*val*): store

If *val* is positive, returns 1; if it is negative, −1; if it is zero, 0.
This function can be included in constant expressions if *val* is constant.

abs(*val*): store

Returns $|val|$.
This function can be included in constant expressions if *val* is constant.

modulus(`x1`, `y1`, `x2`, `y2`): store

Returns the Euclidean distance between (x_1, y_1) and (x_2, y_2).

power(`base`, *[exponent]*): store

Returns $base^{exponent}$. If `exponent` is not given, it defaults to 2.

This function can be included in constant expressions if both parameters are constant.

sin(`angle`, *[divisor]*): store
cos...

Returns the sine or cosine of `angle` (measured in degrees), multiplied by 32640. If `divisor` is given, the return value is divided by it.

angle(`x1`, `y1`, `x2`, `y2`): store

Returns the angle of the line between (x_1, y_1) and (x_2, y_2). The return value is measured in degrees, such that 0° is a line oriented vertically upwards.

index_series(`index`, `offset`, `init`, [{`start`, `end`, `endval`, *[mode]*}...]): store

This function is not available in RealLive prior to 1.2.6.

Returns $f(index + offset)$, where $f()$ is a function I do not fully understand

It is easiest to describe in terms of a graph plotting the values of $y = f(x)$. In the basic case where no ranges $(start, end, endval, mode)$ are provided, $f(x) = init$ for all values of x. If a single range is provided with $mode = 0$, then

$$f(x \leqslant start) = init$$

$$f(x > end) = endval$$

$$f(start < x \leqslant end) = init + (endval - init)\frac{x - start}{end - start}$$

Things get complicated when $mode \neq 0$ and when more than one range is provided. When more than one range is provided, then, roughly speaking, the value of $init$ for each range is taken to be the $endval$ of the previous range.

When $mode$ is 1 or 3, the range describes an accelerating curve, and when $mode$ is 2 or 4 it describes a decelerating curve. I have not attempted to deduce formulae for these yet.

When two ranges overlap, they appear to cancel out in some way; I have not managed to work out precisely what's happening in these cases.

If anyone can figure out the exact function used here, or can write a better explanation, I would be only too glad to hear from them.

As an example:

```
for (int input = 0, entry) (input <= 10) (input += 1):
  entry = index_series(input, 0, 5, {0, 5, 10}, {8, 10, 0})
  itoa_s(input, 2) + ', ' + itoa_s(entry, 2)
  par;
pause
```

produces the output

0, 5
1, 6
2, 7
3, 8
4, 9
5, 10
6, 10
7, 10
8, 10
9, 5
10, 0

6.6.2 Arrays

array? (*symbols...*): integer constant

Intrinsic macro: expands to 1 if all *symbols* are arrays, 0 otherwise.

length (*array*): integer

Returns the length of *array*, which must be an array variable.

6.6.3 Integer blocks

setarray(*origin*, *values...*)
setarray_stepped(*origin*, *step*, *values...*)

Sets a block of integers, starting with *origin*, to the given values. *values* is an arbitrary number of integer expressions, each of which is assigned in turn to the next variable.

Note that despite the functions' names, *origin* may not be an array variable. To fill an array *arr*, pass *arr*[0] for *origin* instead.

With `setarray_stepped()`, *step* is added to the address of the current variable after each assignment to find the next variable to use. A *step* of 1 produces behaviour identical to `setarray()`.

setrng(*array*, *[value]*)
setrng(*first*, *last*, *[value]*)
setrng_stepped(*origin*, *step*, *count*, *[value]*)

Sets a block of integers to *value*. The default value, when none is given, is 0. `setrng()` modifies all variables between *first* and *last* inclusive, or all members of *array*; `setrng_stepped()` modifies *count* variables, taking every *step*th variable from *origin*. In other words, the following commands are equivalent:

```
setrng(arr, 1)
setrng(arr[0], arr[length(arr) − 1], 1)
setrng_stepped(arr[0], 1, length(arr), 1)
```

cpyrng(*source*, *dest*, *count*)

Copies a block of values of length *count* from *source* to *dest*. The function appears to succeed even if the ranges overlap.

cpyvars(*dest*, *offset*, *source*...)

Fills consecutive variables, starting with *dest*, with values drawn from the list *source*; the value read in each case is the variable at the address of the corresponding element in *source* adjusted by the given offset.

The above is probably unclear, but examples should explain adequately:

```
cpyvars(arr[0], 0, a, b, c, d, e, f)
```

is the same as

```
setarray(arr[0], a, b, c, d, e, f)
```

with the exception that *a* etc. must be variables; and

```
cpyvars(arr[3], 2, foo[3], bar[2], baz[1])
```

is equivalent to

```
arr[3] = foo[5]
arr[4] = bar[4]
arr[5] = baz[3]
```

sum(*first*, *last*): store

Returns the sum of the contents of the block of variables between *first* and *last* inclusive.

For example:

```
int arr[10] = 2
x = sum(arr[0], arr[9])  // x = 20
```

sums({*first*, *last*}...): store

Returns the combined sum of all the ranges given.

For example:

```
int arr[10] = 2
x = sum({arr[0], arr[3]}, {arr[4], arr[9]})  // x = 20
```

6.6.4 Strings

strused(*var*): store

Returns 0 if the string variable *var* is empty, otherwise 1.

strlen(*var*): store

Returns the length of *var*. Double-byte characters are counted as two bytes.

strcharlen(*var*): store

Returns the number of characters in *var*. Double-byte characters are counted as one character.

strcpy(*dest*, *source*, *[count]*)

Copies the first *count* characters of *source* into the variable *dest*. *source* may be a string literal or #res directive as well as a string variable. If *count* is omitted, the whole string is copied.

In most circumstances, it is clearer to write

```
s = r + 'foo'
```

than

```
strcpy(s, r + 'foo')
```

strcat(*dest*, *source*)

Appends *source* to the variable *dest*.

As with strcpy(), it is usually clearer and more efficient to write

```
s += r
s += 'foo'
```

than

```
strcat(s, r + 'foo')
```

strclear(*first*, *[last]*)

Empties the given string variable.

If *last* is given, the whole range from *first* to *last* will be emptied. If *first* is an array, *last* must not be given, and the whole array will be emptied.

strtrunc(*var*, *length*)

Truncates *var* such that its length does not exceed *length* characters.

stralloc(*var*, *length*)

Allocates memory for *var* such that its length is greater than or equal to *length* characters. Any existing value is lost. The new value is undefined.

The main purpose of this function is to ensure that extension DLLs can write to string variables safely. There is no way to determine the *capacity* (as distinct from length) of a string: therefore it may be necessary to call this function to ensure that the string has at least the required capacity before calling the DLL, or risk an access violation if it did not.

strsub(*source*, *offset*, *[length]*): string
strrsub...

Returns *length* characters from *offset* in *source*. If *length* is not given, the remainder of the string is returned.

For strsub(), the offset is calculated from the start of the string. For strrsub(), it is counted backwards from the end. In both cases, the substring extends forwards from *offset*, and the count is based on whole characters, not bytes.

To return all characters to the left of a given index, you can use the long form of strcpy() instead.

strpos(*string*, *substring*): store

Returns the offset of the first instance of *substring* in *string*, or -1 if *substring* is not found.

strlpos(*string*, *substring*): store

As strpos(), but returns the offset of the last instance of *substring*. If *substring* appears only once, or not at all, in *string*, the behaviour is identical with that of *strpos*.

strcmp(*a*, *b*): store

Compares *a* and *b*. Returns 0 if they are equivalent, −1 if *a* would be sorted before *b*, and 1 if *b* would be sorted before *a*. The sort order is that of JIS X 0208.

You may prefer to use an expression form rather than this function:

a == b *equals* strcmp(a, b) == 0
a > b *equals* strcmp(a, b) > 0
a <= b *equals* strcmp(a, b) <= 0

and so on.

Uppercase(*[source]*): string
Lowercase...

Modifies the case of ASCII characters in *source*. *source* is optional; if it is not supplied, the current contents of the variable to which the result is assigned are used. That is to say,

```
str foo = Uppercase('foo')
```

and

```
str foo = 'foo'
foo = Uppercase
```

are equivalent.

These functions do not affect full-width letters.

hantozen(*[source]*): string
zentohan...

As Uppercase(), but affecting character width rather than case. hantozen() converts half-width ASCII and katakana characters to their full-width equivalents, and zentohan() performs the opposite transformation.

itoa(*value*, *[length]*): string
itoa_s...
itoa_w...
itoa_ws...

Converts the integer *value* into a decimal representation. The optional parameter *length* specifies a minimum length for the result.

The result is right-aligned and (if shorter than *length*) padded with zeroes (for itoa() and itoa_w()) or spaces (for itoa_s() and itoa_ws()). If *value* is negative, a minus sign is prepended, which is not included in *length*. Finally, if the function called was itoa_w() or itoa_ws(), the return value is converted to full-width characters, so that *length* in this case refers to characters, *not* bytes.

For example, itoa_s(−1, 3) returns '−␣␣1'.

digits(*value*): store

Returns the number of digits in the decimal representation of *value*. This is the length of the string that would be generated by an itoa() call with *length* set to 1, excluding any minus sign.

digit(*value*, *dest*, *n*): store

Sets the variable *dest* to equal the *n*th digit from the right in the decimal representation of *value*, and returns the total number of digits in the number.

digit() can be defined in terms of other functions: for example,

```
x = digit(a, y, b)
```

is equivalent to

```
y = strrsub(itoa(a), b, 1)
x = digits(a)
```

atoi(*string*): store

Returns the value of the integer represented by *string*, or 0 if *string* does not represent an integer. Leading whitespace is ignored, as is anything following the last decimal digit.

intout(*value*)
strout(*string*)

Function forms of the control codes \i and \s. You should use the control codes for preference, as the functions will not be taken into account by Rlc's lineation routine.

6.6.5 Name variables

GetName(*index*): string
GetLocalName...

Returns the $index^{th}$ name variable of the appropriate type. See section 5.4.2 for details.

SetName(*index*, *string*)
SetLocalName...

Sets the $index^{th}$ name variable of the appropriate type to *string*. See section 5.4.2 for details.

6.7 Input

6.7.1 Selections

select(*options*...): store
select_s...

select_w *[window]* (*options...*): store

The `select()` family of functions are the primary interface between player and game in a standard visual-novel-style work. They present a set of options. The return value is a zero-based integer indicating the option selected, or a negative value indicating an error.

The difference between the three variants is the manner in which the options are presented:

- `select()` displays them in the normal text window; unlike the AVG32 equivalent, it appears *not* to break them into columns if there are too many options to fit in the window, so use with care.
- `select_s()` displays them as a series of buttons, formatted according to the settings of the `#SELBTN` variables in `gameexe.ini`. This is the command used to display Fūko's "donna itazura wo?" selections in *Clannad*.
- `select_w()` displays them in a separate selection window, which automatically expands to fit the number of options.

  The window specifier *window* is an integer, enclosed in square brackets, which can optionally come between `select_w()` and its parameter list. This selects the window to use for the options. It must be defined with `#WINDOW`, and `#WINDOW.SELCOM_USE` must be set for it. If *window* is omitted, the value of `#DEFAULT_SEL_WINDOW` is used.

options is a list of strings, with a special extension. Each string can optionally be preceded by an effect specifier, which is a semicolon-delimited list of effects terminated with a colon. Each effect, in turn, is a combination of a function and an optional condition. The recognised functions are:

- hide

  The option is not presented at all, although a return value is still allocated for it. This is probably only meaningful with conditions.
- blank

  A blank, non-selectable line is presented in place of the option.
- colour(*colour*)

  Displays the option in the given colour, which is an index to `#COLOR_TABLE` in `gameexe.ini`.
- title (*[colour]*)

  Disables the option. If *colour* is specified, it has the same effect as the `colour` function as well.
- cursor(*index*)

  Displays an icon next to the option when it is highlighted. The icon chosen is defined by `#CURSOR.`*index* in `gameexe.ini`.

A condition is added by following the function with **if** and then an expression; the effect will be applied only if the expression evaluates to a non-zero value.

For example, you might write a simple main menu like this:

```
repeat
  case
    select_w (
      'New game',
```

```
        title if LatestSave < 0: 'Load game',
        hide if !game_finished: 'Omake',
        'Exit'
      )
    of 0 // 'New game' selected
      jump(game_start_scenario)
    of 1 // 'Load game' selected
      menu_load
      break
    of 2 // 'Omake' selected
      farcall (omake_scenario)
      break
    other // 'Exit' selected (or an error occurred)
      end
    ecase
  till 0
```

ReturnPrevSelect()

Jumps back to the previous `select()`, `select_w()`, or `select_s()` call.

6.7.2 Text boxes

CreateInput(*index*, *x*, *y*, *width*, *height*, *fontsize*, *br*, *bg*, *bb*, *fr*, *fg*, *fb*)

Creates a text input box identified as *index*. The box covers the area (*x*, *y*)-(*x* + *width*, *y* + *height*), has a background colour defined by the RGB triplet (*br*, *bg*, *bb*), and the font it uses is *fontsize* pixels high and coloured (*fr*, *fg*, *fb*).

CloseInput(*index*)
CloseAllInputs()

`CloseInput()` deletes the indexed input box; `CloseAllInputs()` deletes all open input boxes.

FocusInput(*index*)

Sets the keyboard focus to input box *index*.

SetInput(*index*, *text*)

Sets the contents of input box *index* to *text*.

GetInput(*index*): string

Returns the contents of input box *index* as a string.

6.7.3 Mouse input

GetClick(*x*, *y*): store

Waits for a click, then fills the variables *x* and *y* with the current location of the mouse cursor, and returns a value indicating which mouse button was pressed:

1 left button
0 neither
-1 right button

WaitClick(*time*, *x*, *y*): store

A cross between GetClick() and waitC(). Waits up to *time* ms; when the time runs out, or sooner if the player clicks, fills *x* and *y* with the location of the mouse cursor and returns a value as for GetClick().

GetCursorPos(*x*, *y*, *[button1*, *button2]*)

Fills the variables *x* and *y* with the current location of the mouse cursor; if variables *button1* and *button2* are provided, they are additionally filled with the current status of each mouse button.

The following values are used to indicate a button's status:

0 unpressed
1 being pressed
2 pressed and released

FlushClick()

Resets each mouse button's status to 0.

SetCursorPos(*x*, *y*)

Moves the cursor to (*x*, *y*).

6.7.4 Keyboard input

CtrlPressed(): store

Returns 1 if the Control key is pressed, 0 otherwise. RealLive seems to interpret the AltGr key as another Control key.

ShiftPressed(): store

Returns 1 if the Shift key is pressed, 0 otherwise.

CtrlKeySkip(): store
CtrlKeySkipOn()
CtrlKeySkipOff()

If the 'Control-key skip' flag is enabled, holding Control will fast-forward through pauses and text display, even if the text in question has not been viewed previously. CtrlKeySkip() returns the current value of the flag, while the other two functions turn it on and off respectively.

The default can be set in gameexe.ini with the #CTRL_USE variable.

KeyMouseOn()
KeyMouseOff()

Enables or disables the use of the cursor keys to move the mouse pointer.

6.8 Text window controls

6.8.1 Pausing and breaking

pause()

Pause between paragraphs of text, waiting for a mouse click. While the game is paused, the text log can be viewed, text windows hidden, the menu displayed, games saved and loaded, etc.

If `#WINDOW.R_COMMAND_MOD` is set to 0 for the active window, this function behaves identically to `page()`, i.e. the active text window is cleared when the game continues, and all other open text windows are closed. If it is set to 1, `pause()` ends a paragraph rather than a page, inserts a line break instead of clearing the window, and leaves other text windows unmodified.

pause_all()

As `pause()`, but other text windows are never closed when the game continues, regardless of `#WINDOW.R_COMMAND_MOD` settings.

page()

Pause between screens of text: as `pause()`, but when the game continues, the active text window is always cleared, and other windows always closed, regardless of the setting of `#WINDOW.R_COMMAND_MOD`.

spause()

Pause within a paragraph of text: as `pause()`, but when the game continues, the active text window is not cleared, nor is a new line begun, and other open text windows are not affected. This is the function equivalent to the control code `\p`.

br()

Inserts a hard line break, preserving the current indentation level. This is the function equivalent to the control code `\n`.

par()

Inserts a hard line break, and resets indentation to zero. This is the function equivalent to the control code `\r`.

6.8.2 Moving text

TextOffset(x, y)
TextOffsetX(x)
TextOffsetY(y)

Move the text insertion point, i.e. the position relative to the text window at which the next character to be output will be printed, by (x, y) pixels.

These are the function equivalents to the control codes `\mv`, `\mvx`, and `\mvy`; see the descriptions of those for the precise semantics and applications.

TextPos(*x*, *y*
TextPosX(*x*)
TextPosY(*y*)

As TextOffset() etc., except *x* and *y* are absolute rather than relative to the current insertion point.

6.8.3 Appearance

TextWindow(*[index]*)

Changes the active text window. Text windows are defined with the #WINDOW variables in gameexe.ini. If *index* is not given, resets it to the default, usually (always?) 0.

GetWindowPos(*index*, *origin*, *x*, *y*)
GetWindowDefaultPos(*index*, *origin*, *x*, *y*)

Fills the variables *origin*, *x*, and *y* with the current or default position of window *index*.

As with the gameexe.ini setting #WINDOW.POS, *x* and *y* are the absolute distance between an edge of the screen, determined by *origin*, and the nearest edge of the window. Values of *origin* are:

0 top and left
1 top and right
2 bottom and left
3 bottom and right

SetWindowPos(*index*, *origin*, *x*, *y*)
ResetWindowPos(*index*)

Set the position of window *index*, either to the given position (defined as for GetWindowPos()), or to the default defined in gameexe.ini.

GetWakuAll(): store
SetWakuAll(*style*)

Returns or modifies the current window decoration style.

FontColour(*[text]*, *[shadow]*)
SetFontColour...
SetFontColourAll...

Sets the text foreground to the colour of index *text*, and the text shadow to colour *shadow*. If only *text* is given, the default shadow colour is used, and if neither parameter is given, both colours are reset to their defaults.

- FontColour() is equivalent to the control code \c, in that the colours are reset after the next pause() call.
- SetFontColour() modifies the colour permanently.
- SetFontColourAll() does the same for all windows at once.

There is no control code equivalent for either of these latter forms.

FontSize(*[size]*)
FontSizeAll...

Sets the font size to *size*, or to the default if *size* is not given. The behaviour of FontSize() is identical to the control code \size. FontSizeAll() does the same for all windows at once.

GetWindowAttr(*r*, *g*, *b*, *[alpha]*, *[filter]*)
DefWindowAttr...

The parameters are integer variables, which are filled with the values of the window attributes: the RGB components of its background colour, the *alpha* value with which it is composited with the image behind, and a value *filter* which determines the composition mode: if it is 0, a subtractive filter is used, while if it is 1, a straightforward alpha filter is used.

GetWindowAttr() returns the current values, while DefWindowAttr() returns the defaults, as defined in gameexe.ini with the variable #WINDOW_ATTR. (Note that there are also window-specific settings #WINDOW.ATTR, which will override the global settings on a window-by-window basis if the relevant #WINDOW.ATTR_MOD is 1.)

SetWindowAttr(*r*, *g*, *b*, *[alpha]*, *[filter]*)

Modifies window attributes. The parameters have the same meaning as in GetWindowAttr(), except that arbitrary integer expressions may be used.

If no values are supplied for the *alpha* and *filter* parameters, those settings are not modified.

WindowAttrR(): store
WindowAttrG...
WindowAttrB...
WindowAttrA...
WindowAttrF...
DefWindowAttrR...
DefWindowAttrG...
DefWindowAttrB...
DefWindowAttrA...
DefWindowAttrF...

Functions permitting direct access to current and default values of individual window attributes. The code

```
r = WindowAttrR
g = WindowAttrG
b = WindowAttrB
a = WindowAttrA
f = WindowAttrF
```

is equivalent to

```
GetWindowAttr(r, g, b, a, f)
```

SetWindowAttrR(*r*)
SetWindowAttrG(*g*)
SetWindowAttrB(*b*)

SetWindowAttrA(*alpha*)
SetWindowAttrF(*filter*)

Functions permitting direct modifications of individual window attributes.

6.8.4 Text speed

FastText()

Enables 'fast text' mode: text is printed immediately, rather than one letter at a time, regardless of the message speed settings.

NormalText()

Disables 'fast text' mode.

SetMessageNoWait(*flag*)

Sets the internal 'no wait' flag to *flag*. If it is 1, text is printed without pausing between letters; if it is 0, the speed at which text is printed depends on the message speed setting.

Note that this setting is independent of 'fast text' mode. The player can use the 'no wait' flag to force text always to be printed at maximum speed, but they cannot use it to force text always to be printed slowly.

MessageNoWait(): store
DefMessageNoWait...

These return, respectively, the current and the default settings for the internal 'no wait' flag.

SetMessageSpeed(*speed*)

Sets the message speed to *speed*, where 0 is practically instantaneous and 255 is very slow.

MessageSpeed(): store
DefMessageSpeed...

These return, respectively, the current and the default settings for the message speed.

6.8.5 Clearing text windows

msgHide()

Closes the active text window.

msgHideAll()

Closes all open text windows.

msgHideAllTemp()

Hides all open text windows temporarily; they reappear unchanged when the next command operates on them.

msgClear()

Empties, but does not remove, the active text window.

msgClearAll()

Empties, but does not remove, all open text windows.

6.8.6 Window animations

EnableWindowAnm(*window*)
DisableWindowAnm(*window*)

Enable or disable opening and closing animations for *window*. Animations are enabled by default. If they are disabled, windows appear and disappear instantaneously.

GetOpenAnmMod(*window*): store
GetCloseAnmMod...
GetOpenAnmTime...
GetCloseAnmTime...

Retrieve the current settings for window animations. The initial values are defined in `gameexe.ini` with variables such as `#WINDOW.OPEN_ANM_MOD`: see 8.2.3.2 for details.

SetOpenAnmMod(*window*, *mod*)
SetCloseAnmMod...
SetOpenAnmTime(*window*, *ms*)
SetCloseAnmTime...

Modify the current settings for window animations. See 8.2.3.2 for valid parameters and their meanings.

6.8.7 Character portraits

Character portraits are pictures attached to a text window (and therefore shown and hidden with it), typically used to give the expressions of speaking characters. They are ordinary `g00` bitmaps, attached to the window by being opened in a 'face' slot; there are up to eight of these (although most games use one at most), defined in `gameexe.ini` with the `#WINDOW.FACE` variables.

FaceOpen(*file*, *[index]*)

Loads *file* as a character portrait, displaying it as face *index*, or face 0 if *index* is not given.

FaceClear(*[index]*)

Removes face *index*, or face 0 if *index* is not given.

6.9 Sound

6.9.1 Sound settings

SetSoundQuality(*setting*)

Sets the sound quality, i.e. the sample size and frequency to use for all sound output. Valid settings are:

0	11 kHz	8 bit
1	11 kHz	16 bit
2	22 kHz	8 bit
3	22 kHz	16 bit
4	44 kHz	8 bit
5	44 kHz	16 bit
6	48 kHz	8 bit
7	48 hKz	16 bit

SoundQuality(): store

Returns the current sound quality.

SetReduceDistortion(*flag*)
ReduceDistortion(): store

Set or get a flag of unknown meaning. RealLive describes its function as "enable this if you get distortion during sound playback".

6.9.2 Music

Music playback operates on 'tracks', which are predefined in `gameexe.ini` with the `#CDTRACK` and `#DSTRACK` variables.

SetBgmEnabled(*flag*)

Set to 1 to enable music playback, or to 0 to disable it.

BgmEnabled(): store

Returns 1 if music playback is enabled, or 0 otherwise.

SetBgmVolMod(*level*)

Sets the volume of music relative to other sound playback. *level* should be between 0 and 255.

BgmVolMod(): store

Returns the current music volume modifier.

bgmPlay(*track*, *[fadein]*, *[fadeout]*)

Starts *track* playing. If *fadein* is given, the track will fade in over *fadein* ms. If *fadeout* is given, and another track was already playing, that track will simultaneously fade out over *fadeout* ms, otherwise it will be cut off immediately.

The music is played in the background: control passes to the next statement immediately.

bgmPlayEx(*track*, *[fadein]*, *[fadeout]*)

As `bgmPlay()`, but plays in the foreground rather than the background: that is, the call does not return until the track has finished playing.

bgmLoop(*track*, *[fadein]*, *[fadeout]*)

As `bgmPlay()`, but rather than playing the track once, it is played repeatedly: each time it reaches the end, it returns to the loop position given in the track definition.

bgmWait()

If a music track is currently playing, waits for the current playthrough to end and then stops it. The function does not return until the music has stopped. In other words, this function converts a previous `bgmPlay()` or `bgmLoop()` call into a `bgmPlayEx()` call.

bgmStatus(): store

Returns a value indicating the status of the music subsystem:

- **0** Idle
- **1** Playing music
- **2** Fading out music

bgmPlaying(): store

Returns a value indicating whether the music subsystem is playing a track normally. Equivalent to

```
bgmStatus == 1
```

bgmStop()

Stops any music that might have been playing.

bgmFadeOut(*fadetime*)
bgmFadeOutEx(*[fadetime]*)

Stops music by fading it out over *fadetime* ms. `bgmFadeOut()` returns immediately and fades the music out in the background; `bgmFadeOutEx()` pauses the game while it fades and does not return until the music has stopped.

bgmRewind()

If a music track is currently playing, rewinds it to the beginning.

bgmTimer(): store

If a music track is currently playing, returns the current position in the track, in ms, counting from the beginning of the track (not from when the track began playing: that is, the timer is reset each time a looping track restarts).

bgmSetVolume(*level*, *[fadetime]*)

Sets the music volume to *level*, which should be between 0 and 255. The actual volume used will be

$$\frac{level \times mod_{bgm}}{255}$$

where mod_{bgm} is the volume level controlled by `SetBgmVolMod()`.

If *fadetime* is given, the volume will change smoothly, with the change taking *fadetime* ms, otherwise it will change instantly.

bgmVolume(): store

Returns the music volume level. (This is the value equivalent to the input to `bgmSetVolume()`, not the actual output volume, of course.)

bgmMute(*[fadetime]*)
bgmUnMute...

Sets the music volume level to the minimum and maximum respectively. That is to say,

bgmMute(1000) *equals* bgmSetVolume(0, 1000)
bgmUnMute(1000) *equals* bgmSetVolume(255, 1000)

6.9.3 Sound effects

Sound effects are played from standard `wav` files, which are referenced directly by name (sans extension).

To permit multiple sound effects to be played simultaneously, the RealLive system provides a number of channels: 16, plus eight more that can only be accessed indirectly by `wavPlay()`. Most sound effect functions therefore take a parameter *channel*, which should be an integer between 0 and 15. If a new sound effect is ordered on a channel which is already playing one, the previous sound is cancelled, so channels must be allocated carefully.

SetPcmEnabled(*flag*)

Set to 1 to enable wave file playback, or to 0 to disable it.

PcmEnabled(): store

Returns 1 if wave file playback is enabled, or 0 otherwise.

SetPcmVolMod(*level*)

Sets the volume of wave files relative to other sound playback. *level* should be between 0 and 255. This is a global setting affecting all channels.

PcmVolMod(): store

Returns the current wave file volume modifier.

wavPlay(*filename*, *[channel]*, *[fadein]*)

Plays *filename*`.wav`. If *fadein* is given, the sound effect fades in over *fadein* ms.

The sound effect is played in the background; control passes to the next statement immediately.

Uniquely to this function, *channel* is optional. If it is omitted, the sound effect is played back in one of 8 'general-purpose' channels, which are separate from the 16 normal channels. These are controlled automatically, and cannot be operated on by most of the other functions in this section.

wavPlayEx(*filename*, *channel*, *[fadein]*)

As `wavPlay()`, but control does not pass to the next statement until the sound effect has finished playing.

wavLoop(*filename*, *channel*, *[fadein]*)

As `wavPlay()`, but rather than playing the track once, it is played repeatedly.

wavWait(*channel*)

If a sound effect is currently playing in *channel*, waits for the current playthrough to end and then stops it. The function does not return until the sound effect has stopped. In other words, this function converts a previous `wavPlay()` or `wavLoop()` call into a `wavPlayEx()` call.

wavPlaying(*channel*): store

Returns 1 if a sound effect is currently playing in *channel*, or 0 otherwise.

wavStop(*[channel]*)

Stops the sound effect in *channel*, or all sound effects if *channel* is not given.

wavStopAll()

Stops all sound effects.

wavFadeOut(*channel*, *fadetime*)

Stops the sound effect in *channel* by fading it out over *fadetime* ms.

wavRewind(*channel*)

If a sound effect is currently playing in *channel*, rewinds it to the beginning.

wavSetVolume(*channel*, *level*, *[fadetime]*)

Sets the volume level of *channel* to *level*, which should be between 0 and 255. The actual volume used is calculated relative to the overall sound effect volume modifier in the same way as for `bgmSetVolume()`.

If *fadetime* is given, the volume will change smoothly, with the change taking *fadetime* ms, otherwise it will change instantly.

wavVolume(*channel*): store

Returns the volume level of *channel*.

wavMute(*channel*, *[fadetime]*)
wavUnMute...

Sets the volume level of *channel* to the minimum and maximum respectively. That is to say,

wavMute(0) *equals* wavSetVolume(0, 0)
wavUnMute(0) *equals* wavSetVolume(0, 255)

6.9.4 Interface sounds

Interface sounds are, as the name suggests, sound effects normally attached to interface events. As with other sound effects, they are stored in standard `wav` files, but

their manner of invocation is different.

Interface sounds are defined in `gameexe.ini` with the `#SE` variables.

SetSeEnabled(*flag*)

Set to 1 to enable interface sound playback, or to 0 to disable it.

SeEnabled(): store

Returns 1 if interface sound playback is enabled, or 0 otherwise.

SetSeVolMod(*level*)

Sets the volume of interface sound effects relative to other sound playback. *level* should be between 0 and 255.

SeVolMod(): store

Returns the current interface sound effect volume modifier.

sePlay(*index*)

Plays the indexed interface sound.

6.9.5 Voices

SetKoeEnabled(*flag*)

Set to 1 to enable voice playback, or to 0 to disable it.

KoeEnabled(): store

Returns 1 if voice playback is enabled, or 0 otherwise.

SetKoeVolMod(*level*)

Sets the volume of voice relative to other sound playback. *level* should be between 0 and 255.

KoeVolMod(): store

Returns the current voice volume modifier.

SetUseKoe(*character*, *flag*)

Set to 1 to enable voices for *character*, or 0 to disable them. This permits voice control on a per-character basis.

UseKoe(*character*): store

Returns the current value of the UseKoe flag for the given character.

SetKoeMode(*mode*)

Selects a voice playback mode, i.e. which form of communication to use for strings having both text and voice data:

- **0** Text and voice
- **1** Text only
- **2** Voice only

KoeMode(): store

Returns the current voice playback mode.

SetBgmKoeFade(*flag*)

Sets the music/voice fade flag. This determines whether to decrease the music volume while voice data is playing. If *flag* is 1, music is faded; if it is 0, the volume is not modified.

BgmKoeFade(): store
DefBgmKoeFade...

Return the current and default value of the music/voice fade flag.

SetBgmKoeFadeVol(*flag*)

Sets the amount by which the music volume is modified when the music/voice fade flag is active.

BgmKoeFadeVol(): store
DefBgmKoeFadeVol...

Return the current and default values of the music/voice fade volume modifier.

koePlay(*koe*, *[character]*)
koeDoPlay...

Plays voice sample *koe*. If *character* is given, it identifies a speaker for use with the `UseKoe()`-based selective character voice activation. `koePlay()` takes this into account, but `koeDoPlay()` always plays the voice, regardless of UseKoe(*character*).

koePlayEx(*koe*, *[character]*)
koeDoPlayEx...

These are to `koePlay()` and `koeDoPlay()` as `wavPlayEx()` is to `wavPlay()`.

koePlayExC(*koe*, *[character]*): store
koeDoPlayExC...

As `koePlayEx()` and `koeDoPlayEx()`, but can be cancelled with a mouse click. The return value is 1 if this happened, 0 otherwise. (Note that it is 0 if the sample was cancelled by fast-forwarding with Ctrl.)

koeWait()

Pauses until the currently playing voice sample ends. Effectively converts a previous `koePlay()` call into `koePlayEx()`.

koeWaitC(): store

As `koeWait()`, but can be cancelled with a mouse click. The return value is as for `koePlayExC()`.

koePlaying(): store

Returns 1 if a voice sample is currently playing, or 0 otherwise.

koeStop()

Stops the currently playing voice sample, if any.

koeSetVolume(*level*, *[fadetime]*)

Sets the voice volume to *level*, which should be between 0 and 255. The actual volume used is calculated relative to the overall sound effect volume modifier in the same way as for bgmSetVolume().

If *fadetime* is given, the volume will change smoothly, with the change taking *fadetime* ms, otherwise it will change instantly.

koeVolume(): store

Returns the voice volume level.

koeMute(*[fadetime]*)
koeUnMute...

Sets the voice volume level to the minimum and maximum respectively. That is to say,

koeMute(1000) *equals* koeSetVolume(0, 1000)
koeUnMute(1000) *equals* koeSetVolume(255, 1000)

6.10 Graphics

A naming convention is used in this section to streamline the documentation: For every function with a name beginning `rec`, there exists another function with of the same name, with `rec` replaced with `grp`. These pairs of functions all have identical parameters and behaviour, with one exception: where the `rec` function takes parameters of the form "x, y, $width$, $height$", denoting a rectangle from (x, y) to $(x+width, y+height)$, the `grp` function interprets the same parameters as "x, y, x', y'", denoting a rectangle from (x, y) to (x', y').

6.10.1 Screen settings

ScreenWidth(): integer constant
ScreenHeight(): integer constant

These macros expand to the width and height of the screen, in pixels, based on the setting of #SCREENSIZE_MOD found at compile time.

ModeToScreenSize(*mode*, *width*, *height*)

Fills the variables *width* and *height* with the screen size associated with screen mode *mode*. This is a function, not a macro, and is evaluated at runtime.

The purpose of this function is not actually clear. I suppose programmers worried that their code may be executed in a future interpreter that uses different screen sizes could use it like this:

```
int width, height
ModeToScreenSize(gameexe('SCREENSIZE_MOD'), width, height)
```

But it's hard to imagine this being a serious concern. So I'm probably missing something.

6.10.2 Device contexts

Graphics are stored in memory in 'device contexts' (DCs). Most graphics functions take at least one DC as an argument, which is the bitmap that will be affected by the operation. DC 0 is the screen buffer, and changes to it are reflected on screen; it is always allocated. There are then 15 offscreen DCs, which can be used to store, combine, and modify graphics before copying to the screen. DC 1 is initially allocated, and may not be smaller than the screen, but the rest are initially unallocated and may be any size.

DCs do not have to be allocated manually. When a bitmap is loaded, the target DC is automatically freed (if existing) and reallocated to the size of the bitmap. Likewise, if a non-load operation (blitting, drawing, etc.) has an unallocated DC as its target, a bitmap of the same size as the screen is allocated automatically in that DC. The only time a DC needs manual allocation is if you want to blit or draw into an empty DC that is of a different size from the screen, in which case the `allocDC()` function provides the necessary functionality.

The term 'DC', and many of the function names, are carried over from my AVG32 toolkit, where I was forced to come up with my own names for everything. With RealLive, the provision of a debug mode means I know that VisualArt's refer to their offscreen bitmaps as 'banks', and that (for example) the function `grpCMaskCopy()` is called BOXCOPYEASY in their development kit. I have retained my own names out of habit. VisualArt's names are given in appendix A.

allocDC(*dc*, *width*, *height*)

Allocates a blank $width \times height$ bitmap in *dc*. Any DC apart from DC 0 may be allocated thus, although DC 1 is never given a size smaller than the screen resolution. Any previous contents of *dc* are erased.

freeDC(*dc*)

Frees *dc*, releasing the memory allocated to it. DC may not be 0; if it is 1, DC 1 will be blanked, but not released.

Freeing is not necessary to reuse the DC, but it's good manners not to hold onto megabytes unnecessarily, right?

GetDCPixel(*x*, *y*, *dc*, *r*, *g*, *b*)

Fills the variables (*r*, *g*, *b*) with the colour of the pixel at (*x*, *y*) on the given DC.

See `SEEN0002` of the *Maiden Halo* demo for an example of why you might want to do this.

6.10.3 Default bitmaps

The *filename* parameters in bitmap loading functions normally take the name of a `g00` file, but there is also a special filename `'???'`. This refers to one of two "previous bitmap" variables, the "default grp" (which is used in `grp` and `rec` functions), and the "default bgr" (used in `bgr` functions). These are set automatically to the files loaded by certain functions (such as `grpOpenBg()`), but may also be queried and modified with the functions in this section.

DefaultGrp(): string

Returns the default grp filename.

SetDefaultGrp(*filename*)

Modifies the default grp filename.

DefaultBgr(): string

Returns the default bgr filename.

SetDefaultGrp(*filename*)

Modifies the default bgr filename.

6.10.4 Masks

RealLive's bitmaps are 32-bit, i.e. 24-bit RGB colour with an 8-bit alpha channel, which is used by some functions to determine transparency when combining graphics. (The naming convention is that functions which take the alpha channel into account have 'Mask' as the first word of their name, but all cases are also documented explicitly.)

In addition to these, however, it is possible to load masks separately, and some functions permit the specification of an arbitrary mask to use in addition to any internal mask. (Such functions generally have a 'WithMask' suffix on their names, but again, they are documented explicitly.) This section details the functions operating on external masks.

grpLoadMask(*filename*, *mask*)

Loads *filename* into the mask buffer *mask*. *filename* should be a greyscale bitmap.

6.10.5 Simple effects

wipe(*dc*, *r*, *g*, *b*)

Fills *dc* with the colour indicated by the given RGB triplet.

recFade(*[x*, *y*, *width*, *height]*, *colour*, *[time]*)
recFade(*[x*, *y*, *width*, *height]*, *r*, *g*, *b*, *[time]*)

Fades the screen to the given colour, which can be either an index to the #COLOR_TABLE variables in `gameexe.ini` or an RGB triplet. If *time* is given, the fade lasts *time* ms, otherwise it is instantaneous. If an area is given, only that area is affected, otherwise the whole screen is faded.

recFlash(*[x*, *y*, *width*, *height]*, *r*, *g*, *b*, *[time]*)

A simple flash effect: the background is filled with the given colour (an RGB triplet) for *time* ms, or for a brief but unspecified time if *time* is not given. If the first four parameters are given, the flash is confined to the area of the screen they specify.

6.10.6 Loading and displaying bitmaps

recOpen(*filename*, *effect*, *[opacity]*)
recOpen(*filename*, *effect*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*)
recOpen(*filename*, *x*, *y*, *width*, *height*, *dx*, *dy*, *time*, *style*, *direction*,

interpolation, *xsize*, *ysize*, *a*, *b*, *opacity*, *c*)
recMaskOpen...

These functions load and display a single bitmap. *filename* is loaded into DC 1, and then copied to DC 0 with the given *effect* (a reference to the #SEL or #SELR variables in gameexe.ini) and, if *opacity* is given, that degree of opacity.

If the second or third form of the command is used, only the area (x, y)-$(x + width - 1, y + height - 1)$ of the bitmap will be copied, and it will be placed at the position (dx, dy) on the target.

If the third form is used, rather than reading *effect* from #SEL.*effect*, the transition effect used is defined by the parameters *time*, *style*, *direction*, *interpolation*, *xsize*, *ysize*, *a*, *b*, *opacity*, and *c*. These have the same meanings as in the #SEL definitions.

grpMaskOpen and recMaskOpen are the same as grpOpen and recOpen, except that *filename*'s alpha mask is used to determine transparent areas in the loaded bitmap.

recOpenBg(*filename*, *effect*, *[opacity]*)
recOpenBg(*filename*, *effect*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*)
recOpenBg(*filename*, *x*, *y*, *width*, *height*, *dx*, *dy*, *time*, *style*, *direction*, *interpolation*, *xsize*, *ysize*, *a*, *b*, *opacity*, *c*)

As recOpen(), but with two additional effects.

Firstly, the default grp filename will be set to *filename*.

Secondly, the foreground layer is cleared, and anything in the background layer is promoted to the foreground layer: this applies to both SerialPdt buffers (section 6.11.2) and object buffers (section 6.12).

recLoad(*filename*, *dc*, *[opacity]*)
recLoad(*filename*, *dc*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*)
recMaskLoad...

Loads *filename* into *dc*; note that *filename* may *not* be '???'. If the second form is used, the given area of the bitmap is loaded at the given location.

recMaskLoad is the same as recLoad, except that *filename*'s alpha mask is used to determine transparent areas in the loaded bitmap.

grpBuffer(*filename*, *dc*, *[opacity]*)
grpBuffer(*filename*, *dc*, *x*, *y*, *x'*, *y'*, *dx*, *dy*, *[opacity]*)
grpMaskBuffer...

As grpLoad() and grpMaskLoad(). Despite my best efforts, I have not been able to determine how, if at all, these functions are different from the ...Load versions, except that there are no rec equivalents.

recDisplay(*dc*, *effect*, *[opacity]*)
recDisplay(*dc*, *effect*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*)
recDisplay(*dc*, *x*, *y*, *width*, *height*, *dx*, *dy*, *time*, *style*, *direction*, *interpolation*, *xsize*, *ysize*, *a*, *b*, *opacity*, *c*)

As recOpenBg(), but taking the source from the DC *dc* instead of from a file. The behaviour is otherwise identical to recOpenBg(). In particular, the fore-

ground layer is cleared, and the background layer is promoted to take its place.

AVG2000: the `opacity` *parameter only exists in the third form.*

recMulti(*source*, *effect*, *[opacity]*, *compositors*...)
recMulti(*source*, *effect*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*, *compositors*...)
recMulti(*source*, *x*, *y*, *width*, *height*, *dx*, *dy*, *time*, *style*, *direction*, *interpolation*, *xsize*, *ysize*, *a*, *b*, *opacity*, *c*, *compositors*...)

Loads multiple bitmaps and composes them automatically before displaying them. *source* is loaded into DC 1, then the *compositors* are processed in order, and finally the final bitmap is copied to DC 0 with the transition *effect*, *opacity* if given, etc., as with `recOpenBg()`. Also as with `recOpenBg()`, the foreground layer is cleared and the background layer is promoted to take its place.

source can be either a filename or the index of an existing DC. In the former case, it replaces the default grp filename.

compositors is any number of compositors, which are special functions valid only in `grpMulti()` and `recMulti()` calls. The following compositors exist:

copy(*filename*, *[effect]*, *[opacity]*)

Loads *filename* and adds it to the image, using its alpha mask for transparent areas, and using *opacity* if given for the overall transparency level. If *effect* is given, its x, y, x', y', dx, and dy values will be used to select an area of the bitmap and position it on the output.

area(*filename*, *x*, *y*, *width*, *height*, *dx*, *dy*, *[opacity]*)

As `copy()`, but with the position and target position specified in full, instead of being read optionally from an effect definition. This behaves in the same way as `grpMaskBuffer()`.

As an example, the command

```
grpMulti('background', 12,
        copy('char1', 0, 0, 319, 479, 0, 0),
        copy('char2', 0, 0, 319, 479, 320, 0))
```

is equivalent to

```
grpLoad('background', 1)
grpMaskBuffer('char1', 1, 0, 0, 319, 479, 0, 0)
grpMaskBuffer('char2', 1, 0, 0, 319, 479, 320, 0)
grpDisplay(1, 12)
```

6.10.7 Blitting

recCopy(*src*, *dst*, *[opacity]*)
recCopy(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*, *[opacity]*)
recMaskCopy...

Copies DC *src* to DC *dst*.

If the second form is used, the area (x, y)-$(x+width, y+height)$ — or the absolute equivalent, for grpCopy (see the naming convention explanation under section 6.10) — is copied to (dx, dy), otherwise the whole bitmap is copied to (0, 0).

If *opacity* is given, the copied area will be blended with that level of transparency.

recMaskCopy is the same as recCopy, except that the source bitmap's alpha mask is used to determine transparent areas in the copied area.

recMaskBlend(*src*, *dst*)
recMaskBlend(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*)

The purpose of this function is uncertain. It appears to be exactly the same as recMaskCopy(), except that the optional *opacity* parameter in the latter function is not available at all.

recCopyWithMask(*src*, *dst*, *mask*, *[opacity]*)
recCopyWithMask(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*, *mask*, *[maskXmod*, *maskYmod*, *levels*, *threshold]*, *[opacity]*)
recCopyInvMask...
recMaskCopyWithMask...
recMaskCopyInvMask...

As recCopy() and recMaskCopy(), but additionally using an external mask (see section 6.10.4), selected with the *mask* parameter.

The additional optional parameters *maskXmod*, *maskYmod*, *levels*, and *threshold* affect the external mask. It is shifted by the first two before use: for example, if the mask were 640 × 480 pixels, and contained a gradient fill in the form of a single cycle of a sine wave, then setting *maskXmod* to −160 would cause it to be used in the form of a single cycle of a cosine wave. It is reduced to *levels* uniform shades of grey with a standard 'nearest match' method. Finally, if *threshold* is non-zero, each level α in the mask is transformed such that

$$\alpha' = \frac{\alpha \cdot threshold}{256}$$

A common technique is to use a frame counter to advance *threshold* from −256 to 256 while repeatedly copying the image: this can be used to perform arbitrarily shaped wipes.

recCopyInvMask is the same as recCopyWithMask, except that the external mask is inverted. The same is true for recMaskCopyInvMask.

recSwap(*src*, *dst*)
recSwap(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*)

Swaps the given area of DC *src* with an area of the same size on DC *dst*.

recStretchBlt(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dwidth*, *dheight*, *dst*, *[opacity]*)
recMaskStretchBlt...

As recCopy() and recMaskCopy(), except the copied area is deformed so that it fits into the given area on DC *dst*.

As in all cases, the grp variants use absolute coordinates to define areas, including the destination area.

recRotate(*x*, *y*, *width*, *height*, *xorg*, *yorg*, *src*, *dx*, *dy*, *dwidth*, *dheight*, *dxorg*, *dyorg*, *angle*, *xscale*, *yscale*, *opacity*)

recMaskRotate...

This set of functions is not available in RealLive prior to 1.1.5.

Takes a rectangular area of DC `src`, scales it by (x_{scale}, y_{scale}) (which are percentages), and rotates it $\frac{angle}{10}^{o}$ clockwise around the point (x_{org}, y_{org}). It is then blitted on to DC `dst`, being positioned such that (x_{org}, y_{org}) is placed at (dx_{org}, dy_{org}), and cropped such that only areas falling within (dx, dy)-$(dx + dwidth - 1, dy + dheight - 1)$ are modified. The `opacity` parameter is used in the same way as that of `recCopy()`.

As in all cases, the `Mask` variant causes the source bitmap's alpha mask to be used to determine transparent areas in the blitted bitmap, and the `grp` variants use absolute coordinates to define both the source and the destination areas.

recCMaskCopy(`src`, `dst`, `r`, `g`, `b`, *[*`opacity`*]*)

Copies the contents of DC `src` to DC `dst`. The colour specified by the RGB triplet (`r`, `g`, `b`) is used as a mask to define transparent areas in the copied image. Note that the normal alpha mask appears to be used *in addition* to this colour mask.

6.10.8 Filters

recOutline(*[*`x`, `y`, `width`, `height`*]*, `dc`, `r`, `g`, `b`, *[*`opacity`*]*)

Draws a box around the given area of `dc` (or the whole bitmap, if no area is given), in the colour (`r`, `g`, `b`).

recFill (*[*`x`, `y`, `width`, `height`*]*, `dc`, `r`, `g`, `b`, *[*`opacity`*]*)

As `recOutline()`, but the box drawn is filled with the same colour.

recInvert(*[*`x`, `y`, `width`, `height`*]*, `dc`, *[*`opacity`*]*)

Inverts the colours in the given area. If `opacity` is given, it is used to combine the inverted version with the original. Care should be taken, as an opacity of 128 effectively fills the area with solid grey.

recMono(*[*`x`, `y`, `width`, `height`*]*, `dc`, *[*`opacity`*]*)

Converts the given area to greyscale.

recColour(*[*`x`, `y`, `width`, `height`*]*, `dc`, `r`, `g`, `b`)

Applies a solid colour to the given area. Positive values are applied with a 'screen' operation, negative values with a 'multiply' operation. The effect is to tint the image.

recLight(*[*`x`, `y`, `width`, `height`*]*, `dc`, `level`)

Applies a solid greyscale of the given `level` to the given area. If `level` is positive, it is applied with a 'screen' operation; if `level` is negative, it is applied with a 'multiply' operation. The effect is to brighten or darken the image.

This function is, in practice, identical to `recColour()` with the colour set to (`level`, `level`, `level`).

6.10.9 Filtered blits

recAnd(*src*, *dst*)
recAnd(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*)
recOr...

Copies from DC *src* to DC *dst*, applying the result with an 'and' or 'or' operation.

recAdd(*src*, *dst*, *[opacity]*)
recAdd(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*, *[opacity]*)
recSub...
recMaskAdd...
recMaskSub...

As `recCopy()` and `recMaskCopy()`, except that the results are combined differently: the `Add` versions are applied with an 'add' operation, and the `Sub` versions first invert the copied bitmap and then apply it with a 'subtract' operation.

recAddWithMask(*src*, *dst*, *mask*, *[opacity]*)
recAddWithMask(*x*, *y*, *width*, *height*, *src*, *dx*, *dy*, *dst*, *mask*, *[maskXmod*, *maskYmod*, *levels*, *threshold]*, *[opacity]*)
recSubWithMask...
recMaskAddWithMask...
recMaskSubWithMask...
recAddInvMask...
recSubInvMask...
recMaskAddInvMask...
recMaskSubInvMask...

As `recCopyWithMask()`, etc., but using the same combining modes as `recAdd()`, etc.

recRotateAdd(*x*, *y*, *width*, *height*, *xorg*, *yorg*, *src*, *dx*, *dy*, *dwidth*, *dheight*, *dxorg*, *dyorg*, *angle*, *xscale*, *yscale*, *opacity*)
recRotateSub...
recMaskRotateAdd...
recMaskRotateSub...

As `recRotate()`, etc., but using the same combining modes as `recAdd()`, etc.

6.10.10 Zooming and scrolling

recZoom(*x1*, *y1*, *width1*, *height1*, *x2*, *y2*, *width2*, *height2*, *src*, *dx*, *dy*, *dwidth*, *dheight*, *time*)

Zooms from the first area to the second area of DC *src*, copying each frame into the destination area on DC 0, deforming it to fit if necessary; the whole operation is adjusted to last *time* ms. Control does not pass to the next function until the operation has completed.

The precise operation can be illustrated by duplicating its functionality with other functions:

recZoom(0, 0, 640, 480, 160, 120, 320, 240, 1, 0, 0, 640, 480, 2000)

zooms in to a 2× scale on the centre of DC 1, over two seconds, and is equivalent to

```
int x, y, w, h
InitFrames ({10, 0, 160, 2000}, {12, 640, 320, 2000},
            {11, 0, 120, 2000}, {13, 480, 240, 2000})
while ReadFrames ({10, x}, {11, y}, {12, w}, {13, h})
  recStretchBlt (x, y, w, h, 1, 0, 0, 640, 480, 0)
```

recPan(*x1*, *y1*, *x2*, *y2*, *src*, *dx*, *dy*, *width*, *height*, *time*)

Similar to recZoom(), except that all areas copied are the same size, which is defined by the size of the destination window. That is to say,

```
recPan(x1, y1, x2, y2, src,
       dx, dy, width, height, time)
```

is equivalent to

```
recZoom(x1, y1, width, height, x2, y2, width, height, src,
        dx, dy, width, height, time)
```

recShift(*x*, *y*, *dim_x*, *dim_y*, *src*, *dx*, *dy*, *width*, *height*, *direction*, *time*)
recSlide...

Scrolls an area of DC *src* in. The scrolling takes place within the window defined by *dx*, *dy*, *width*, and *height* of DC 0, and lasts *time* ms. The direction of the scrolling is determined by *direction*, which has the following values:

0	down
1	up
2	right
3	left

The exact area that is scrolled in depends on the direction. If the scrolling is horizontal, then the block that is scrolled in will be $dim_x \times height$ pixels; if the scrolling is vertical, it will be $width \times dim_y$ pixels. Either way, it is taken from (*x*, *y*), and scrolled in until it is just fully visible.

The difference between recShift and recSlide is that the former scrolls the previous contents of the window out as the new contents are scrolled in, while the latter scrolls the new contents in over the old contents.

6.10.11 Displaying text and numbers

grpTextout(*text*, *x*, *y*, *dst*, *size*, *r*, *g*, *b*)

Prints the text *text* to DC *dst*, with the top left of the first character being at (*x*, *y*). The font used is the system font used for all normal text display. *size* determines the size of characters, and the triplet (*r*, *g*, *b*) determines the text colour.

grpNumber(*n*, *digits*, *pad*, *sign*, *x*, *y*, *width*, *height*, *mod_x*, *mod_y*, *src*, *dx*, *dy*, *mod_dx*, *mod_dy*, *dst*, *[opacity]*)
grpMaskNumber...

Displays the number `n`, using bitmapped digits.

If `pad` is 1, it is padded with zeroes to `digits` digits first; if `pad` is 0, `digits` is ignored.

If `sign` is 1, a plus is prepended to positive numbers; if it is 0, no plus is prepended, but a minus is still added to negative numbers. (To add a space before positive numbers, so that the size of the output will be constant, set `sign` to 1 and leave the plus bitmap blank.)

The digits are copied from DC `src`. They should be arranged such that each digit is an area of $width \times height$ pixels, with the bitmap for 0 located at (x, y), and the offset from each digit to the next being (mod_x, mod_y): that is, such that a digit d is located at $(x + d \cdot mod_x, y + d \cdot mod_y)$. The bitmap for the minus sign is treated as digit 10, and the bitmap for the plus sign is digit 11.

Digits are printed to DC `dst`, starting at (dx, dy), and incrementing the location by (mod_{dx}, mod_{dy}) after each digit.

If `opacity` is given, the copy operation will use that value for the overall transparency of each digit. If the command used is grpMaskNumber, the source bitmap's alpha mask will be used to determine transparent areas in the digits.

6.10.12 Haikei and bgr functions

There are a number of functions which operate on 'haikei' (backgrounds); *Alma* and *Realize* use them. I have not been able to work out what many of them do, so for now they are undocumented. Of them, only the functions bgrLoadHaikei (which appears to be similar to `grpOpen()`, but without modifying any DCs) and bgrMulti (which appears to be very similar to `grpMulti()`) are given names at present.

6.10.13 The graphics stack

RealLive stores details of the most recent few graphical operations on a stack, possibly to enable it to reconstruct the scene when loading a saved game. It can be viewed by selecting 'Graphics stack' from the 'Window' menu in debug mode. These functions permit you to modify its contents directly.

stackSize(): store

Returns the current length of the stack.

stackClear()

Empties the stack.

stackPop(`count`)

Removes up to `count` entries from the stack.

stackTrunc(`length`)

Truncates the stack by discarding all but the oldest `length` entries. That is, stackTrunc(len) is equivalent to stackPop(stackSize − len).

stackNop(`count`)

Adds *count* NOP commands to the stack. Why you would ever want to do this is unclear.

6.10.14 Controlling screen updates

refresh()

The `refresh()` function yields control from the bytecode interpreter to the RealLive system proper, permitting screen updates to take place.

This function must always be called to update the screen when the drawing mode is set to manual with `DrawManual()`. It is also necessary to call this regardless of drawing mode in tight loops, if the screen is changed other than by blitting to DC 0. See 6.13.3 for an example of this latter usage.

DrawAuto()

Enables automatic refreshes: the screen is redrawn after every load or blit to DC 0.

DrawSemiAuto()

Enables semi-automatic refreshes. The difference from automatic refreshes is not clear.

DrawManual()

Disables automatic refreshes. The screen is only updated by explicit `refresh()` calls. This can be used to get smooth animation even when rendering directly to DC 0.

6.11 Animations

6.11.1 Basic effects

shake(*spec*)

Shakes the screen, using the movements defined in `gameexe.ini` with the variable `#SHAKE`.*spec*. This function appears to have been included for compatibility with AVG32; equivalents with more functionality are `ShakeSpec()` and `ShakeLayersSpec()`.

6.11.1.1 Simple shaking

ShakeScreen(*direction*, *amount*, *speed*, *rep*, *faderep*)
ShakeScreenEx...

Shakes the screen with a predefined effect. The *amount* and *speed* paramters define how far and fast to move; this movement will be repeated *rep* times to its full extent, and then *faderep* times more with the exaggeration of the effect being reduced each time. (If both *rep* and *faderep* are 0, the bouncing will continue without end until another shake effect is applied or `ShakeStop()` is called.)

The *direction* parameter determines which effect to use, and takes one of the following values:

DOWNUP	Shake vertically in both directions
RIGHTLEFT	Shake horizontally in both directions
UP	Bounce up
DOWN	Bounce down
LEFT	Bounce left
RIGHT	Bounce right
ZOOM	Shake 'towards' player by zooming

ShakeScreen() returns immediately and shakes in the background; ShakeScreenEx() does not return until the effect is complete.

ShakeScreen2D(*hamount*, *hspeed*, *vamount*, *vspeed*, *rep*, *faderep*)
ShakeScreen2DEx...

Similar to ShakeScreen() and ShakeScreenEx(), but applying the RIGHTLEFT and DOWNUP effects at the same time; the former is controlled by the *hamount* and *hspeed* paramters, the latter by *vamount* and *vspeed*.

ShakeSpec(*spec*, *rep*, *faderep*)
ShakeSpecEx...

Shakes the screen, using the movements defined in gameexe.ini with the variable #SHAKE.*spec*. The *rep* and *faderep* parameters have the same meaning as for ShakeScreen().

ShakeStop(*[time]*)

Cancels the effect of any active ShakeScreen(), ShakeScreen2D(), or ShakeSpec() call, either by decreasing the extent of the shaking to 0 over *time* ms, or immediately if *time* is not given.

6.11.1.2 Layer-based shaking

ShakeLayers(*direction*, *amount*, *speed*, *rep*, *faderep*, *win*, *txt*, *bg*, *objs*, *[flag]*)
ShakeLayersEx...

As ShakeScreen(), but only certain elements of the screen are affected, depending on which parameters are set. The following elements can be affected by passing a non-zero value in the corresponding parameter:

win	Text window backgrounds
txt	The text in the windows
bg	The background graphics
objs	Objects

The *flag* parameter is not understood; it is optional, and appears to have the same effect as *objs*.

The *direction*, *amount*, *speed*, *rep*, and *faderep* parameters have the same meanings as for ShakeScreen(), with one exception: the *direction* value ZOOM is not supported by ShakeLayers().

ShakeLayers() returns immediately and shakes in the background; ShakeLayersEx() does not return until the effect is complete.

ShakeLayers2D(*hamount*, *hspeed*, *vamount*, *vspeed*, *rep*, *faderep*, *win*, *txt*, *bg*, *objs*, *[flag]*)
ShakeLayers2DEx...

Similar to `ShakeLayers()` and `ShakeLayersEx()`, but applying the `RIGHTLEFT` and `DOWNUP` effects at the same time; the former is controlled by the *hamount* and *hspeed* paramters, the latter by *vamount* and *vspeed*.

ShakeLayersSpec(*spec*, *rep*, *faderep*, *win*, *txt*, *bg*, *objs*, *[flag]*)
ShakeLayersSpecEx...

Shakes selected elements of the screen, using the movements defined in `gameexe.ini` with the variable `#SHAKE`.*spec*. The remaining parameters have the same meaning as for `ShakeLayers()`.

ShakeLayersStop(*[time]*, *[flag]*)

Causes any active layer shake effect to stop. If *time* is specified, the shake is faded out over *time* ms; otherwise it stops immediately. The meaning of *flag* is unclear.

6.11.2 'SerialPdt' animation

'SerialPdt' is the VisualArt's name for animation produced using ordinary bitmaps, as opposed to dedicated animation or video files.

Up to 512 'SerialPdt' commands can be defined at a time, but you will have to allocate buffers for them (the *buf* parameters) individually. There are 256 buffers, each of which has a 'foreground' and a 'background' slot. Creating an animation in the foreground causes it to display immediately, while background animations are displayed only when promoted to the foreground with a function that performs such a layer promotion, such as `grpOpenBg()`.

A naming convention is used to streamline the documentation of this section. Each function documented with a `snm` prefix assigns the animation it defines to the foreground slot of buffer *buf*; for each such function there also exists a version with the prefix `snmBg`, which is identical except that it assigns the animation it defines to the background slot.

6.11.2.1 Frame-based animation

snmPlay(*buf*, *x*, *y*, {*filename*, *time*}...)

Displays each named file in turn, placing each at (*x*, *y*) on DC 0 and showing it for *time* ms.

The animation plays in the background, and control is passed to the next command immediately.

snmPlayEx(*buf*, *x*, *y*, {*filename*, *time*}...)

As `snmPlay()`, but control does not pass to the next command until the animation has finished.

snmLoop(*buf*, *x*, *y*, {*filename*, *time*}...)

As `snmPlay()`, but the animation is played repeatedly, looping each time it finishes, until it is stopped or another command is given the same value of *buf*.

snmPlayCmp(*buf*, *x*, *y*, {*filename*, *time*}...)
snmPlayCmpEx...
snmLoopCmp...

These functions appear to do the same thing as `snmPlay()` etc.; there is no obvious difference in their behaviour.

However, the debugging information indicates that a value 'Mod' for each *filename* is 'Compress' for the Cmp functions, whereas it is 'Thaw' (i.e. decompress) for the versions without Cmp. The meaning of this is unclear. Could it have something to do with whether graphics are decompressed in advance or as needed, or something along those lines?

snmPlayNc(*buf*, *x*, *y*, {*filename*, *time*}...)
snmPlayNcEx...
snmLoopNc...

These functions appear to do the same thing as `snmPlay()` etc.; there is no obvious difference in their behaviour.

However, no debugging information is shown for individual *filename*s with these versions. The reason why is unclear - could it be that files are not cached in advance at all with these versions, or something along those lines?

snmStretch(*buf*, *x1*, *y1*, *x2*, *y2*, {*filename*, *time*}...)
snmStretchEx...
snmStretchLoop...
snmStretchCmp...
snmStretchCmpEx...
snmStretchLoopCmp...
snmStretchNc...
snmStretchNcEx...
snmStretchLoopNc...

As `snmPlay()` etc., except the animation is stretched and/or distorted such that each frame fills the window (x_1, y_1) to (x_2, y_2).

6.11.2.2 Scrolling animation

snmScroll(*buf*, *x1*, *y1*, *x2*, *y2*, *filename*, *right*, *down*, *left*, *up*, *time*)

Scrolls the bitmap *filename* across the area of DC 0 with corners (x_1, y_1) and (x_2, y_2). The bitmap is placed at (x_1, y_1), being tiled to fill the window if necessary, and then scrolled according to the parameters *right*, *down*, *left*, and *up* over *time* ms. It is not deleted between steps, so it will leave a trail if it has a mask.

The distance scrolled is $right - left$ horizontally and $down - up$ vertically, constrained such that it is never greater than the relevant dimension of the scrolled bitmap.

snmScrollEx(*buf*, *x1*, *y1*, *x2*, *y2*, *filename*, *right*, *down*, *left*, *up*, *time*)

As `snmScroll()`, but control does not pass to the next command until the animation has finished.

snmScrollLoop(*buf*, *x1*, *y1*, *x2*, *y2*, *filename*, *right*, *down*, *left*, *up*, *time*)

As snmScroll(), but the animation is played repeatedly, looping each time it finishes, until it is stopped or another command is given the same value of *buf*.

By specifying the scrolled image's dimensions for scroll distance, it is possible to use this command to mimic the infinitely-scrolling backgrounds seen in the *Tokimeki Memorial* series. But please don't.

6.11.3 Videos

RealLive can play back video files in several formats, including AVI and MPG.

movPlay(*file*, *x1*, *y1*, *x2*, *y2*)

Plays a video stream from *file*. It is played within the area (x_1, y_1) to (x_2, y_2), being scaled to fit if necessary.

The video plays in the background, and control is passed to the next command immediately.

movPlayEx(*file*, *x1*, *y1*, *x2*, *y2*)
movPlayExC...: store

As movPlay(), but control does not pass to the next command until the video has finished. Videos played by movPlayEx() cannot be cancelled; videos played by movPlayExC() can be cancelled by clicking the mouse, and the return value of the function is non-zero if this was the case and zero if the video finished normally.

movLoop(*file*, *x1*, *y1*, *x2*, *y2*)

As movPlay(), but the video is played repeatedly, looping each time it finishes, until it is stopped or another video is played.

movStop()

Stops any background video playback.

movWait()

If a video effect is currently playing, waits for the current playthrough to end and then stops it. The function does not return until the video has stopped. In other words, this function converts a previous movPlay() or movLoop() call into a movPlayEx() call.

movPlaying(): store

Returns 1 if a video is currently playing, or 0 otherwise.

6.12 Objects

Objects are retained graphical elements, modifiable separately from the main DCs and automatically composited with the screen.

Up to 512 objects can be defined at a time, but you will have to allocate buffers for them (the *buf* parameters) individually. There are 256 buffers, each of which has a 'foreground' and a 'background' slot; creating an object in the foreground causes

it to display immediately, while background objects are displayed only when promoted to the foreground manually or with a function that performs such a layer promotion, such as `grpOpenBg()`.

Rendering of visible objects takes place in numerical order, so where two objects overlap, that with the lower buffer number appears behind that with the higher.

A naming convention is used to streamline the documentation of this section. Each function documented with a `obj` prefix operates on the foreground slot of object buffer *buf*; for each such function there also exists a version with the prefix `objBg`, which is identical except that it operates on the object in the background slot.

The functions in the sections 'Object position' and 'Object attributes' (section 6.12.3) and onwards) extend this convention: these functions also have variants with prefixes `objRange` and `objBgRange`, which take two parameters *min* and *max* in place of the single *buf* parameter. Such functions apply the transformation described to all objects in the range *min* to *max* inclusive, instead of to the single object *buf*.

6.12.1 Initialising objects

Most of the functions in this section share certain common optional parameters:

- If *visible* is 1, the object will be displayed straight away; if it is 0 or not given, the object will initially be hidden.
- If *x* and *y* are given, the object will be placed at the given location, otherwise at (0, 0).
- *scrollX* and *scrollY* affect the behaviour of the object when scrolling functions such as `ShakeLayers()` are used. It appears that objects are treated as part of the background when scrolling in the X or Y axis if the respective values are non-zero, and as part of the object plane if the values are zero or not given.

objOfArea(*buf*, *[x*, *y*, *x′*, *y′]*, *[visible]*)
objOfRect(*buf*, *[[x*, *y]*, *[width*, *height]]*, *[visible]*)

Creates an empty rectangular 'filter'-type object, filling the entire screen, or the given area (defined with absolute coordinates in `objOfArea()`, and with position and size in `objOfRect()`) if one is given.

Note an idiosyncracy in `objOfRect()`: the *x* and *y* parameters are 'more optional' than *width* and *height*. The relationship between the meaning of the parameters and their number is as follows:

{– 1 –} objOfRect (buf)
{– 3 –} objOfRect (buf, width, height)
{– 4 –} objOfRect (buf, width, height, visible)
{– 5 –} objOfRect (buf, x, y, width, height)
{– 6 –} objOfRect (buf, x, y, width, height, visible)

In the cases where *x* and *y* are not supplied, the object is centered.

The object is initally transparent; applying an effect with a function such as `objLight()` causes it to act as a filter modifying the graphics beneath it.

objOfFile(*buf*, *filename*, *[visible]*, *[x*, *y]*, *[pattern]*, *[scrollX*, *scrollY]*)

Sets object *buf* to hold the bitmap *filename*.

If *pattern* is given, it will use the given bitmap from the source file; this enables you to use a single g00 file containing multiple bitmaps for an object, such as a button, that can have multiple states. The pattern can be changed after object creation with the objPattNo() function.

objOfFileGan(*buf*, *filename*, *ganname*, *[visible]*, *[x*, *y]*, *[pattern]*, *[scrollX*, *scrollY]*)

Similar to objOfFile(), but takes an additional parameter *ganname* which identifies a gan animation file to associate with the object. Animations from this file can then be applied to it (see section 6.12.5 for details).

This function exists only in RealLive. The AVG2000 equivalent is called objOfFileAnm, and does not have the *pattern*, *scrollX*, or *scrollY* parameters; the animation files it uses are in a different format and have the extension anm.

6.12.1.1 Text objects

It is possible to create objects automatically from arbitrary strings. The said string will be displayed in the same font as ordinary display text.

Strings used in text objects are parsed specially according to a unique syntax not used anywhere else in RealLive. Certain sequences of # followed by certain alphabetical characters are interpreted as control codes. The mapping between RealLive text object control codes and Kepago control codes is as follows:

##	Separator
#c	\c
#d	\n
#s	\size
#x	\posx
#y	\posy

The codes #c, #s, #x, and #y can take parameters. When one of these codes is encountered, the following text is scanned to see whether it represents an integer (i.e. matches the POSIX extended regular expression "-?[0-9]+"; if it does, then that integer is passed as a parameter to the code. To prevent following characters being interpreted as parameters, the separator ## may be used, such that the following are equivalent:

```
jOfText(0, '#s10##123#s##456')
jOfText(0, '\size{10}123\size{}456')
```

As the above implies, Rlc automatically converts Kepago control codes encountered within inline strings in calls to objOfText() into this syntax, but it cannot guess whether other string variables will be used in such calls, so you will have to use the #-syntax when constructing strings separately. Kprl never attempts to convert the #-syntax into standard Kepago control codes. So while the two lines of the previous example are equivalent, the following are *not* equivalent:

```
r s = 'foo\nbar' // compilation error – \n not recognised in this context
r s = 'foo#dbar' // compiles happily
jOfText(0, s)    // equivalent to objOfText(0, 'foo\nbar')
```

There is no obvious way to display these special sequences literally. Note also that these codes are not case-sensitive, which doubles the set of clashes without increasing the expressivity of the feature. If all this appears confusing, blame the authors of RealLive, who appear to have delighted in introducing a different ad-hoc string syntax for every construct.

objOfText(*buf*, *text*, *[visible]*, *[x*, *y]*, *[scrollX*, *scrollY]*)

Creates an object consisting of the text *text*. The size, colour, and so forth are normally set with a separate call to `objTextOpts()`. See above for special considerations involved in displaying text in this way.

objSetText(*buf*, *[text]*)
objRangeSetText(*min*, *max*, *[text]*)

Alters the text associated with the given text object(s) to *text*, or to the empty string if *text* is not given; if any objects are not already text objects, they are not modified.

objTextOpts(*buf*, *size*, *xspace*, *yspace*, *vertical*, *colour*, *shadow*)
objRangeTextOpts(*min*, *max*, ...)

Modifies a range of options for the given object(s); if any objects are not text objects, they are not modified.

size	Font size (pixels)
xspace	Extra character spacing (line spacing when vertical)
yspace	Extra line spacing (character spacing when vertical)
vertical	If 1, text is vertical (see note below); if 0, horizontal.
colour	Foreground colour, as an index to `#COLOR_TABLE`.
shadow	Shadow colour, as an index to `#COLOR_TABLE`, or −1 to disable the shadow.

Note that 'vertical' text is not in fact laid out properly in objects; it is simply produced by adding a line break after every full-width character, or every two half-width characters. Japanese punctuation marks are *not* rotated appropriately.

6.12.1.2 Number objects

objOfDigits(*buf*, *filename*, *[visible]*, *[x*, *y]*, *[scrollX*, *scrollY]*)

An object-based equivalent to the `grpNumber()` functions: creates a number object from the given file. *filename* must be a format 2 G00 bitmap, containing 13 patterns: in order, these should be the digits 0 to 9, then the symbols +, −, and ±. The object displays a number using these patterns for its digits.

The object is initially empty; its value must be set separately with a call to `objSetDigits()`.

objSetDigits(*buf*, *value*)
objRangeSetDigits(*first*, *last*, *value*)

Sets the number to be displayed in the given object(s) to *value*.

objNumOpts(*buf*, *digits*, *zero*, *sign*, *pack*, *space*)
objRangeNumOpts(*first*, *last*, ...)

Sets formatting options for the given object(s).

digits	Minimum number of digits; the number is padded to this length.
zero	If 0, padding is with spaces; if 1, with zeroes.
sign	If 0, only negative numbers are signed, and the sign is counted as a digit when present. If 1, all numbers are signed, and the sign is not counted as a digit for padding purposes. The ± pattern is used to sign 0 in this case.
pack	If 1, and the number is less than *digits* digits long, the sign is placed next to the first digit (after the padding); if 0, the sign is placed in the first column and the padding follows it. This has no effect if *zero* is 1.
space	If non-zero, the width in pixels of each digit. If zero, the width is determined by the width of the digit's bitmap instead.

6.12.1.3 Environment objects

objDriftOfFile(*buf*, *filename*, *[visible]*, *[x*, *y]*, *[scrollX*, *scrollY]*)

Creates an environment object from the given file. Environment objects create multiple sprites which drift around or fall at random; they are typically used to simulate weather effects and the like.

objDriftOpts(*buf*, *count*, <?>, *pattern*, <?>, <?>, *yspeed*, <?>, <?>, <drift?>, <spread?>, *driftspeed*, *x*, *y*, x', y')
objRangeDriftOpts(*min*, *max*, ...)
objFadeOpts(*buf*, *min_alpha*, *max_alpha*, <?>, <?>, <?>)
objRangeFadeOpts(*min*, *max*, ...)

Functions modifying certain aspects of environment objects. The precise meaning of most of the options that can be modified is currently unknown, so for now these functions are basically undocumented.

6.12.2 Object management

objDelete(*buf*, *[maxbuf]*)

Deletes object *buf*, or (if *maxbuf* is given) all objects from *buf* to *maxbuf* inclusive.

objClear(*buf*, *[maxbuf]*)

Deletes object *buf*, and clears any position data and attributes. If *maxbuf* is give, all objects from *buf* to *maxbuf* inclusive are affected.

objCopy(*src*, *dst*)

Makes a copy of object *src* in buffer *dst*. All attributes are copied exactly.

objCopyToBg(*src*, *dst*)
objBgCopyToFg...

As `objCopy()`, except that the target layer is different: `objCopyToBg()` copies the foreground object *src* to the background buffer *dst*, and vice versa.

These functions are exceptions to the object naming convention; that is, there is no objBgCopyToBg() (this function is simply `objBgCopy()`), and likewise no objCopyToFg() (this function is simply `objCopy()`).

objWipeCopyOn(*buf*, *[maxbuf]*)
objWipeCopyOff...

Set or clear the 'wipe copy' flag for object *buf*, or, if *maxbuf* is given, all objects from *buf* to *maxbuf* inclusive.

If the 'wipe copy' flag is set, then the object in question, if a foreground object, will not be deleted when the background layer is promoted by a call to `grpDisplay()`, unless there is another object in the corresponding background slot. If the flag is cleared, foreground objects will always be deleted upon layer promotion.

6.12.3 Object position

As documented at the start of section 6.12, each function described in this section actually represents *four* functions: the variants described here affect single objects on the foreground layer, but in addition to the background layer equivalent (replace `obj` with `objBg`), there are also equivalents operating on ranges of objects (`objRange` and `objBgRange`).

objShow(*buf*, *flag*)

Sets the visibility of the given object to *flag*, where 0 means 'invisible' and 1 means 'visible'.

objMove(*buf*, *x*, *y*)
objLeft(*buf*, *x*)
objTop(*buf*, *y*)

Moves the given object to (*x*, *y*); modifies only one coordinate if one of the latter forms is used.

objAdjust(*buf*, *index*, *x*, *y*)
objAdjustX(*buf*, *index*, *x*)
objAdjustY(*buf*, *index*, *y*)

Sets the adjustment position *index* to (*x*, *y*). There are eight adjustment positions per object (numbered 0 to 7); their values are summed and added to the object's base position (as set with `objMove()`) to determine the actual position at which it is displayed.

objAdjustAll(*buf*, *x*, *y*)
objAdjustAllX(*buf*, *x*)
objAdjustAllY(*buf*, *y*)

As `objAdjust()`, etc., except that all eight adjustment positions for the object are set to the same values. It would appear that the main use of this would be to pass 0 for *x* and *y* in order to clear all adjustments.

objAdjustVert(*buf*, *y*)

Similar to `objAdjustY()`, but operating on a separate setting (named 'Height' in the RealLive debugger).

objOrigin(*buf*, *x*, *y*)
objOriginX(*buf*, *x*)
objOriginY(*buf*, *y*)

Sets the origin of the given object; this is the point relative to which it is positioned, and around which rotation, scaling, etc. take place. By default it is (0, 0), i.e. the top left of the object's bounding box.

For example, to rotate a 100 × 200 object 180° and place it at the centre of the screen, one would use the code

```
objOrigin(obj, 50, 100)
objMove(obj, ScreenWidth / 2, ScreenHeight / 2)
objRotate(obj, 1800)
```

where, without the `objOrigin()` call, the object would have been positioned with its top left at the centre of the screen, and then rotated around that point rather than its centre.

objRepOrigin(*buf*, *x*, *y*)
objRepOriginX(*buf*, *x*)
objRepOriginY(*buf*, *y*)

Not fully understood; these functions *appear* to have the same effect as `objOrigin()`, etc., except controlling a second origin setting which is added to the main origin when rotating and scaling an object, but ignored for the purpose of positioning it.

objScale(*buf*, *width*, *height*)
objWidth(*buf*, *width*)
objHeight(*buf*, *height*)

Stretch or shrink the given object; *width* and *height* are percentages of its natural size.

objRotate(*buf*, *angle*)

Sets the angle of the given object to $\frac{angle}{10}^\circ$.

6.12.4 Object attributes

As documented at the start of section 6.12, each function described in this section actually represents *four* functions: the variants described here affect single objects on the foreground layer, but in addition to the background layer equivalent (replace `obj` with `objBg`), there are also equivalents operating on ranges of objects (`objRange` and `objBgRange`).

objPattNo(*buf*, *index*)

For objects based on `g00` bitmaps, selects a bitmap from the file to use for the object. This enables you to use a single `g00` file containing multiple bitmaps for objects, such as buttons, that can have multiple states.

objAlpha(*buf*, *alpha*)

Sets the opacity of the given object to `alpha`, where 0 is completely transparent and 255 is completely opaque.[1]

objDispArea(*buf*, *[x*, *y*, *x′*, *y′]*)
objDispRect(*buf*, *[x*, *y*, *width*, *height]*)

Clips the given object so that only those parts of it that fall within the window defined by the given coordinates are displayed. If the window coordinates are not given, the entire screen is used (that is, the clipping is removed).

objDispCorner(*buf*, *[x*, *y]*)

As `objDispArea()`, but the clipping window is defined as (0, 0)-(*x*, *y*).

objMono(*buf*, *level*)

Converts the given object to greyscale; `level` controls the extent to which colour is removed. The effect is the same as that used by the `grpMono()` function, but the effect can be removed from the object by calling the function again with a `level` of 0.

If the object is a filter (defined with `objOfArea()` or `objOfRect()`), rather than modify the object directly, this causes the object to become a monochrome filter affecting the image beneath it.

objInvert(*buf*, *level*)

Inverts the given object, using `level` to determine how opaquely to combine the inverted form with the original: a `level` of 0 removes the invert effect, while a level of 255 causes it to be completely inverted.

If the object is a filter, rather than modify the object directly, this causes the object to become an invert filter affecting the image beneath it.

objLight(*buf*, *level*)

Applies the same brightness adjustments as `grpLight()` to the object; as with other object functions, calls are not cumulative, so setting `level` to 0 will remove the effect again, and if the object is a filter, it will apply the effect to the parts of the image beneath it. Note that it is both legal and meaningful for `level` to be negative.

objTint(*buf*, *r*, *g*, *b*)
objTintR(*buf*, *r*)
objTintG(*buf*, *g*)
objTintB(*buf*, *b*)

Applies a colour to the object with the same effect as `grpColour()`; the latter three forms permit modification of one channel at a time. The same rules apply as for `objLight()`.

objColour(*buf*, *r*, *g*, *b*, *level*)
objColR(*buf*, *r*)
objColG(*buf*, *g*)

[1] Note that the value is taken modulo 256, so a value of 256 will in fact be transparent again.

objColB(*buf*, *b*)
objColLevel(*buf*, *level*)

Applies a colour to the object by blending it directly at *level* opacity; setting *level* to 0 removes the effect, setting it to 255 causes the object to be shaded solidly with the given colour. The latter four forms permit modification of one parameter at a time.

objComposite(*buf*, *mode*)

Modifies the composition mode used to display the given object:

0	normal	(object is displayed as though with `grpMaskCopy()`)
1	add	(object is displayed as though with `grpMaskAdd()`)
2	subtract	(object is displayed as though with `grpMaskSub()`)

objLayer(*buf*, *layer*)
objDepth(*buf*, *depth*)
objOrder(*buf*, *order*)
objQuarterView(*buf*, *qview*)

Modify some settings for the given object. The meanings of the settings are unknown, since they do not have any obvious effect, other than to alter the values displayed for 'layer', 'depth', 'order', and 'quaterview' *(sic)* in the RealLive debugger.

objScrollRate(*buf*, *x*, *y*)
objScrollRateX(*buf*, *x*)
objScrollRateY(*buf*, *y*)

Alter the 'scroll rate' settings for the given object. These are the same as the *scrollX* and *scrollY* settings that can be set in most object initialisation functions, and determine whether an object is considered part of the background plane or the object plane for scrolling and shaking animations.

6.12.5 Object animations

These functions animate objects based on the patterns in their associated bitmap files.

With the exception of `objStop()`, each of these functions comes in two forms: one with the usual `obj` and `objBg` prefixes, and one with the prefixes `gan` and `ganBg`. The former is used with regular objects, and employs a simple animation based on displaying the patterns of the object in turn, showing each for the duration defined by the function's *time* parameter. The latter is used with objects which were associated with `GAN` animations by initialising with `objOfFileGan()`; in these, an animation ('set') is selected from the `GAN` file with the function's *set* parameter. `GAN` animations provide more flexibility than the simple animations, but are more complicated to set up.[2]

objStop(*buf*, *[pattern]*)

[2] This is an oversimplification. In theory, both sets of functions can be used on objects with and without `GAN`s, with slightly differing effects. In practice, existing RealLive games make a consistent distinction, as documented here.

Stops any animation playing in object *buf*, and sets it to display the given pattern instead; if *pattern* is not given, it defaults to 0.

ganPlay(*buf*, *set*)
objPlay(*buf*, *time*)

Plays an animation. The function returns immediately, leaving the animation playing. When the animation is complete, the object is left displaying the final frame.

ganPlayEx(*buf*, *set*)
objPlayEx(*buf*, *time*)

As previous, but the function does not return until the animation has finished playing.

ganPlayOnce(*buf*, *set*)
objPlayOnce(*buf*, *time*)

Plays an animation, as `ganPlay()`/`objPlay()`; *buf* is cleared when the animation ends.

ganPlayOnceEx(*buf*, *set*)
objPlayOnceEx(*buf*, *time*)

As previous, but the function does not return until the animation has finished playing.

ganLoop(*buf*, *set*)
objLoop(*buf*, *time*)

Plays an animation continuously, looping whenever it reaches the end.

ganPlayBlink(*buf*, *set*)
objPlayBlink(*buf*, *time*)

Plays an animation in a way suitable for animating a character's blinking eyes. A loop is set up in which the animation is played, separated by pauses of approximately three seconds.

In the case of `objPlayBlink()`, the animation displays the patterns of the object first in order, then in reverse order, to enable the closing and opening of an eye to be displayed without duplicating frames. This is *not* the case for `ganPlayBlink()`; it is assumed that the reverse phase will be included in the `GAN` definition.

6.13 Timing

The RealLive system provides three main timing constructs. Where the duration from start to end of a pause is important, use *waiting* functions. Where the duration from the start of a section of code to the end of the pause that ends it is important (but the length of the pause itself is not), use *timers*. Where the rate of change of some attribute is important, use *frame counters*. These constructs are covered in turn in the remainder of this section.

6.13.1 Waiting

wait(*time*)

Pauses for *time* ms. Within a string, the \wait control code has an identical effect.

waitC(*time*): store

Pauses for up to *time* ms; the pause can be cancelled with a mouse click. Valid return values are:

0	The pause was not cancelled, or it was cancelled by use of the Control key.
1	The pause was cancelled with a left click.
-1	The pause was cancelled with a right click.

6.13.2 Timers

Timers are simple counters: when initialised, they start counting from 0, and can be queried at any point to return the number of milliseconds that have elapsed since their initialisation.

RealLive provides two sets of 255 timers, the basic counters ('COUNT') and the extended counters ('EX-COUNT'); the difference between them is unclear, other than that different sets of functions operate on each. Of the functions documented below, those with an 'Ex' infix operate on the extended counters, and those without operate on the basic counters.

The number of the timer to use, *counter*, is optional in all these functions; in all cases, if it is not given, it will default to 0.

All timers run continuously; they cannot be stopped. The implementation appears to be based on the time since the interpreter was started, as this is what you get if you query a timer that has not been initialised to any other value. "Setting" a timer merely stores the time at which the timer was set, and future queries subtract this from the time since the interpreter was started and then add the value to which the timer was initialised.[3]

ResetTimer(*[counter]*)
ResetExTimer...

Sets *counter* to 0.

SetTimer(*time*, *[counter]*)
SetExTimer...

Sets *counter* to *time*.

time(*time*, *[counter]*)
timeEx...

Pauses until *counter* reaches *time*, or returns immediately if *counter* is already greater than *time*.

[3] The timer displays in RealLive's debug mode are only updated when a timer has been accessed within the last twenty seconds, but this is merely a simplification for the programmer's convenience, not a reflection of which are "counting" and which are not.

timeC(*time*, *[counter]*): store
timeExC...

As `time()`, except that the pause can be cancelled with a mouse click. The return value is as follows:

0	The pause was not cancelled.
1	The pause was cancelled with a left click.
-1	The pause was cancelled with a right click.

Timer(*[counter]*): store
ExTimer...

Returns the current value of *counter*.

CmpTimer(*time*, *[counter]*): store
CmpExTimer...

Compares the value of *counter* to *time*, returning 1 if *counter* is greater than *time*, or 0 otherwise.

6.13.3 Frame counters

Frame counters are designed to make it simple to ensure events happen at a constant speed regardless of the host system's specifications. Once a frame counter has been initialised, it will count from one arbitrary number to another, over a given length of time. The counter can be queried at any point to get its current value.

For example, to move a sprite across the screen in exactly five seconds, as smoothly as possible, you might use the following code:

```
objOfFile(2, 'foo', 1, 0, 200) // initialise object 2 from foo.g00
InitFrame(10, 0, 640, 5000)    // set frame 10 to count from 0 to 640
repeat
  int x = ReadFrame(10)        // read value from frame counter
  objLeft(2, x)                // move object
  refresh                      // force a screen redraw
till x >= 640
```

Note that the `refresh()` call is necessary here regardless of the drawing mode selected, since object modifications do not trigger screen refreshes even in automatic mode.

As with timers, RealLive provides two sets of 255 frame counters, the basic frames ('FRAME') and the extended frames ('EX-FRAME'); the difference between them is unclear, other than that different sets of functions operate on each. Of the functions documented below, those with an 'ExFrame' infix operate on the extended frames, and those with just 'Frame' operate on the basic frames.

InitFrame(*counter*, *limit1*, *limit2*, *time*)
InitExFrame...

Starts frame counter *counter* ticking from *limit1* to *limit2*, over a total of *time* ms: that is, the counter will change every

$$\frac{time}{limit_2 - limit_1} \text{ ms}$$

When it reaches *limit2*, it stops.

InitFrameAccel(*counter*, *limit1*, *limit2*, *time*)
InitFrameDecel...
InitExFrameAccel...
InitExFrameDecel...

As `InitFrame()`, but the counter does not change at a constant speed: with *Accel*, it starts slowly and speeds up, while *Decel* has the opposite behaviour.

InitFrameLoop(*counter*, *limit1*, *limit2*, *time*)
InitFrameTurn...
InitExFrameLoop...
InitExFrameTurn...

As `InitFrame()`, but the counter will loop continuously. With *Loop*, it starts from *limit1* each cycle; with *Turn*, it reverses direction each cycle.

InitFrames({*counter*, *limit1*, *limit2*, *time*}...)
InitFramesAccel...
InitFramesDecel...
InitFramesLoop...
InitFramesTurn...
InitExFrames...
InitExFramesAccel...
InitExFramesDecel...
InitExFramesLoop...
InitExFramesTurn...

Equivalents of `InitFrame()`, etc., operating on more than one frame counter at a time. For example,

```
InitExFramesTurn({0, 0, 1000, 2500}, {1, 1000, 0, 2500})
```

is equivalent to

```
InitExFrameTurn(0, 0, 1000, 2500)
InitExFrameTurn(1, 1000, 0, 2500)
```

ReadFrame(*counter*): store
ReadExFrame...

Returns the current value of frame counter *counter*.

ReadFrames({*counter*, *value*}...): store
ReadExFrames...

Returns the current values of multiple frame counters at once. Each *counter* is stored in the corresponding *value* variable. The return value of the whole function is 1 if any of the counters referenced was active, or 0 otherwise. For example, the code

```
active = ReadFrames({100, a}, {101, b}, {102, c})
```

is a more efficient equivalent of

```
a = ReadFrame(100)
b = ReadFrame(101)
```

```
c = ReadFrame(102)
active = FrameActive(100) | FrameActive(101) | FrameActive(102)
```

ClearFrame(*counter*, *[value]*)
ClearExFrame...

Stops *counter* and resets its counter to *value*, or 0 if *value* is not given.

ClearAllFrames(*[value]*)
ClearAllExFrames...

Stops all frame counters and resets their counters to *value*, or 0 if *value* is not given.

FrameActive(*counter*): store
ExFrameActive...

Returns 1 if frame counter *counter* is currently active, or 0 otherwise.

AnyFrameActive: store
AnyExFrameActive...

Returns 1 if any frame counter is currently active, or 0 if all are stopped.

6.14 System functions

6.14.1 Calling extension DLLs

The DLL extension system is documented in 5.6.

The precise availability of the DLL functions is unclear. These functions were introduced somewhere between RealLive 1.1.7.7 and 1.2.3.5, and the ability to load multiple DLLs at once was introduced somewhere between 1.2.3.5 and 1.2.5.5. In the present API, it is assumed - perhaps incorrectly - that the changes take place at the 1.2 and 1.2.5 boundaries.

When used with versions not supporting DLLs at all, the functions always return 0, and a compile-time warning is issued. When used with versions supporting only one DLL, the *index* parameter to each function is ignored, and a compile-time warning is issued if it is non-zero.

LoadDLL(*index*, *filename*): store

This function is not available in RealLive after 1.3.2.

Loads *filename*`.dll` into DLL slot *index*; returns the value returned by the DLL's OnLoad() function, or 0 if the load was unsuccessful.

If there was already a DLL loaded in slot *index*, it will be unloaded first, as though `UnloadDLL()` had been called.

In versions supporting multiple DLLs, you can also load DLLs automatically by naming them with `#DLL` directives in `gameexe.ini`. This becomes the only way to use DLLs from version 1.3.2 onwards.

UnloadDLL(*index*): store

This function is not available in RealLive after 1.3.2.

Unloads any DLL loaded in slot *index*; returns the value returned by the DLL's OnFree() function, or 0 if there was no DLL in that slot.

CallDLL(*index*, *[arg1*, *[arg2*, *[arg3*, *[arg4*, *[arg5]]]]*): store

Calls the DLL loaded in slot *index*; returns the result of calling the DLL's OnCall() function with the given arguments (any not supplied are set to 0), or 0 if there was no DLL in that slot.

6.14.2 Calling external programs

shell(*filename*, *[argv]*)

Passes *filename* to the Explorer shell. If *filename* is a document, it will be opened; if it is an executable, it will be executed with the parameters *argv*. *filename* is relative to `gameexe.ini`.

This appears to be a wrapper to the Win32 ShellExecute() function.

launch(*target*)

Passes *target* to the Explorer shell. Typically it is a URL, which will be opened in the user's default browser.

6.14.3 Time and date

GetDateTime(*y*, *m*, *d*, *wd*, *hh*, *mm*, *ss*, *ms*)

Fills the variables provided with the current time and date: *y*-*m*-*d* *hh*:*mm*:*ss*.*ms*, with *wd* being the day of the week (counting Sunday as 0).

GetDate(*y*, *m*, *d*, *wd*)
GetTime(*hh*, *mm*, *ss*, *ms*)

As `GetDateTime()`, but only parts of the data are returned.

6.14.4 Window settings

title (*text*)

Sets the game title. This is used in two ways: firstly, it is included in saved game names; secondly, if the `gameexe.ini` variable `#SUBTITLE` is non-zero, it will be displayed in the title bar of the game window.

PauseCursor(*index*)

Selects the icon to display in the text window when waiting for input. These are defined in `gameexe.ini` with the `#CURSOR` variables.

MouseCursor(*index*)

Selects a mouse cursor. These are defined in `gameexe.ini` with the `#MOUSE_CURSOR` variables.

GetMouseCursor(): store

Returns the index of the currently selected mouse cursor.

ShowCursor()
HideCursor()

Shows or hides the mouse cursor.

SetCursorMono(*flag*)

Sets the 'monochrome cursor' flag to *flag*. If it is non-zero, the mouse cursor will be reduced to black and white.

This function is provided to improve performance on machines with slow video cards, and "black and white" is a literal description. The greyscale cursor effect in *Clannad*'s dreamworld is achieved by switching to a second cursor style with `MouseCursor()`.

CursorMono(): store

Returns the current value of the 'monochrome cursor' flag.

SetScreenMode(*mode*)

Selects a screen mode. 1 is windowed, 0 is full-screen.

ScreenMode(): store

Returns the current screen mode.

6.14.5 Saved games

savemenu(): store
loadmenu(): store

Display a list of saved games, and return a value indicating which the player selected, or −1 if they clicked 'cancel' instead. `savemenu()` permits any slot to be chosen; `loadmenu()` only permits a slot containing existing save data to be chosen.

save(*slot*)
load...
save_always...
load_always...

Save the current position to *slot*, or load the data from it. In the former case, the title of the saved game is taken from the most recent `title()` call.

`save()` and `load()` optionally display a confirmation dialog (see `ConfirmSaveLoad()`); `save_always()` and `load_always()` never ask for confirmation.

menu_save(): store
menu_load(): store
menu_save_always(): store
menu_load_always(): store

Display a menu and process the result in one step:

menu_save *equals* save(savemenu)
menu_load *equals* load(loadmenu)
menu_save_always *equals* save_always(savemenu)
menu_load_always *equals* load_always(loadmenu)

SaveInfo(*slot*, *y*, *m*, *d*, *wd*, *hh*, *mm*, *ss*, *ms*, *title*): store

If *slot* is empty, returns 0 and leaves the variables in the rest of the parameters untouched. If *slot* contains a saved game, returns 1 and fills the variables with information about it: integers for the date and time it was saved, down to the millisecond (*y*-*m*-*d* *hh*:*mm*:*ss*.*ms*), with *wd* being the day of the week (Sunday being 0), and the value of the `title()` call active when the save was made as a string in *title*.

SaveDate(*slot*, *y*, *m*, *d*, *wd*): store
SaveTime(*slot*, *hh*, *mm*, *ss*, *ms*): store
SaveDateTime(*slot*, *y*, *m*, *d*, *wd*, *hh*, *mm*, *ss*, *ms*): store

As `SaveInfo()`, but only returning parts of the data.

GetSaveFlag(*slot*, {*src*, *dst*, *count*}...): store

Retrieves the values of variables from saved games. If *slot* is empty, returns 0 and does nothing further; if *slot* contains a saved game, returns 1 and processes the list of structures. For each entry in the list, *count* values are copied to a block of variables starting with *dst*, reading from *src*: the values copied are those that are stored in the saved game in *slot*.

For example, an RPG that stored the player's level in F[100], the player's hit points in F[101], and the name of the player's class in S[10], could retrieve these values from saved games to display them in a custom load menu as follows:

```
str menu_line[10]
for (int i = 0) (i < length(menu_line)) (i += 1):
  int (block) level, hp
  str class
  GetSaveFlag(i, {intF[100], level, 2}, {strS[10], class, 1})
  menu_line[i] = 'Level \i{level} \s{class}, \i{hp} HP';
```

LatestSave(): store

Returns the slot most recently saved to, or −1 if no games have been saved.

SetConfirmSaveLoad(*flag*)

Enables or disables the confirmation dialog displayed when saving or loading a game.

ConfirmSaveLoad(): store

Returns 1 if the confirmation dialog is enabled, 0 otherwise.

6.14.6 System command menu functions

The functions in this section operate on system menu commands. Refer to section 5.5 for a list of these commands and their effects.

ContextMenu()

Displays the system command menu, as though the player had right-clicked. Note that if a `#CANCELCALL_MOD` hook is defined, this function has no effect; it neither displays the popup menu nor jumps to the `#CANCELCALL` scenario.

SyscomEnabled(*syscom*): store

Returns 0 if the given system command is invisible, 1 if it is visible, and 2 if it is visible but disabled (greyed out).

HideSyscom(*[syscom]*)

Hides the given system command (sets its state to 0). If *syscom* is not given, the menu as a whole is disabled.

EnableSyscom(*[syscom]*)

Shows and enables the given system command (sets its state to 1). If *syscom* is not given, the menu as a whole is enabled.

DisableSyscom(*syscom*)

Greys out the given system command (sets its state to 2). Note that *syscom* is not optional this time.

InvokeSyscom(*syscom*, *[value]*)

If there is a standard dialog box associated with *syscom*, it is displayed; if there is a standard action, it is performed. The list of menu commands in section 5.5 has details of which menu commands have standard dialogs. The optional *value* is used for the setting where relevant (for example, InvokeSyscom(5, val) is exactly equivalent to SetScreenMode(val)).

ReadSyscom(*syscom*): store

Returns the value associated with the setting *syscom*, where relevant. For example, ReadSyscom(5) is exactly equivalent to `ScreenMode()`.

6.14.7 Menu mode

These functions only seem to work properly when called while in 'menu mode', i.e. while the game itself is suspended due to a `pause()` call, but bytecode is being executed either because a right click has been intercepted by `#CANCELCALL` settings, or because a click on a window button has been intercepted by `#WBCALL` settings.

ShowBackground()

Hide the text window, wait for a click, then show it again and return to the previous paused state.

SetSkipMode()
ClearSkipMode()

Enter or leave 'skip mode', which fast-forwards through previously viewed text.

These functions affect the main game, not the menu system. To modify the skip mode settings from within the game itself, use `SetLocalSkipMode()` and `ClearLocalSkipMode()`.

SkipMode(): store

Returns a value indicating whether 'skip mode' is currently enabled.

This function reads the setting for the main game, not the menu system. To determine whether skip mode is enabled from within the game itself, use `LocalSkipMode()`.

6.14.8 Skip mode

SetLocalSkipMode()
ClearLocalSkipMode()

Enter or leave 'skip mode', which fast-forwards through previously viewed text.

These functions are for use within the game itself. To modify the settings from within a custom menu system, use `SetSkipMode()` and `ClearSkipMode()`.

LocalSkipMode(): store

Returns a value indicating whether 'skip mode' is currently enabled.

This function is for use within the game itself. To read this setting from within a custom menu system, use `SkipMode()` instead.

EnableSkipMode()
DisableSkipMode()

Control whether skip mode can be entered manually from within the standard right-click menu. Regardless of this setting, it will be unavailable if the text on screen has not previously been viewed.

6.14.9 Auto mode

This feature is not available in RealLive prior to 1.2.

It is possible to set the interpreter up to advance text automatically instead of waiting for player input after each screen is displayed; the 'auto mode' controls permit this behaviour to be customised.

When auto mode is disabled, `pause()`, `spause()`, and other related commands pause the game permanently. When it is enabled, they cause the game to pause for a certain length of time: this is calculated as $t_{base} + t_{char} \times n_{chars}$, where n_{chars} is the number of characters that have been printed in the current screen since the last pause.

SetAutoMode(*flag*)

If *flag* is 1, enables auto mode; if it is 0, disables it.

SetAutoBaseTime(*time*)

Sets t_{base} to *time* ms.

SetAutoCharTime(*time*)

Sets t_{char} to *time* ms.

AutoMode(): store
DefAutoMode...

Return the current or default setting of the 'auto mode' flag.

AutoBaseTime(): store
DefAutoBaseTime...

Return the current or default setting of t_{base}.

AutoCharTime(): store
DefAutoCharTime...

Return the current or default setting of t_{char}.

6.14.10 CG mode

Most *bishōjo* games provide a CG mode where special event graphics can be viewed. The RealLive system provides automatic functions for tracking which such graphics have been encountered in the course of gameplay.

The functions rely on the contents of the file defined in `gameexe.ini` with the `#CGTABLE_FILENAME` variable. This file contains a list of graphics designated as CGs, and associates each with a variable, always(?) in the Z[] array, which is set to 1 when the associated file is accessed.[4]

cgGetTotal(): store

Returns the total number of images designated as CGs.

cgGetViewed(): store

Returns the number of CG images that have been viewed.

cgGetViewedPcnt(): store

Returns the percentage of CG images that have been viewed: that is,

cgGetViewedPcnt

is equivalent to

percent(cgGetViewed, cgGetTotal)

cgGetFlag(*filename*): store

Returns the index, usually (always?) in Z[], of the variable associated with *filename*, or −1 if *filename* is not a CG image.

cgStatus(*filename*): store

Returns a value indicating whether *filename* is a CG that has been viewed:

1	CG has been viewed
0	CG has not been viewed
-1	*filename* is not a CG image

This works by reading the associated flag: that is,

[4] The precise format of this file is currently beyond the scope of this manual; a tool to access and modify its contents may be included in a future release of RLdev.

cg = cgStatus('foo')

is usually (always?) equivalent to

cg = cgGetFlag('foo')
if cg >= 0, cg = intZ[cg]

6.14.11 Font settings

SetFontQuality(*value*)
FontQuality(): store

Set or query the antialiasing level for text output. Possible values are:

- **0** No antialiasing
- **1** Low quality
- **2** High quality

SetFontWeight(*value*)
FontWeight(): store

Set or query the font-weight flag. If this is 0, text is rendered normally; if it is 1, text is rendered at a heavier weight. Typically the player will select this in the font selection dialog to compensate for light fonts.

SetFontShadow(*value*)
FontShadow(): store

If the font shadow flag is non-zero, text in the text window is rendered with a shadow.

6.14.12 Miscellaneous flags

SetLowPriority(*flag*)
LowPriority(): store

This flag is described in the default menu as "make this program run slower so that other programs will run smoothly". Its effect is unclear; it does not lower the process priority, but it might cause RealLive to yield control to other processes more frequently.

SetSkipAnimations(*flag*)
SkipAnimations(): store

If this flag is set, animated transitions are skipped.

SetShowObject1(*flag*)
SetShowObject2...
ShowObject1(): store
ShowObject2...

The 'show object' flags are used to provide a way of enabling or disabling interface elements from the menu. If an object's 'ObjectOnOff' property is set to 1 or 2, it will be shown or hidden depending on the corresponding 'show object' flag. This is one of the properties controlled by the #OBJECT variables in gameexe.ini.

In *Clannad*, ShowObject1 is used to control display of the date marker at the top left of the screen (object 84).

SetShowWeather(*flag*)
ShowWeather(): store

The 'show weather' flag determines whether object environment effects are displayed.

SetClassifyText(*flag*)
ClassifyText(): store

The 'classify text' flag apparently has something to do with text colouring, but at the moment I'm not entirely sure what.

SetGeneric1(*value*)
SetGeneric2...
Generic1(): store
Generic2...
DefGeneric1...
DefGeneric2...

RealLive provides two generic settings to permit games using the standard system command menu to include custom options in it. The meaning of each generic flag is left up to the programmer. Valid *value*s are 0 to 4.

6.15 Debugging

RealLive contains a number of functions which spring into action in debug mode (enabled with `#MEMORY` in `gameexe.ini`) to aid the programmer.

Outside debug mode, none of these functions have any effect; you cannot use them to display messages or get input from the player in normal gameplay.

When releasing a program, you can eliminate these functions from the compiled code altogether by passing the `-g` option to Rlc, or by defining the symbol `__NoDebug__`.

__Memory?(): store
__Debugging?(): store

Functions to determine at runtime whether debugging is active. `__Memory?()` returns 1 if `#MEMORY` is set to 1; `__Debugging?()` returns 1 if the 'Debug (F2)' option in the 'Debug' menu is checked (which implies that `#MEMORY` is also 1).

assert(*expr*, *[message]*)

Asserts that the expression *expr* evaluates to a non-zero value; an error is raised if it does not. If *message* is given, it must be a constant string, which will be included in the error message.

If *expr* can be evaluated at compile-time, this is done, any error is raised by the compiler directly, and no code is generated; otherwise a runtime check is compiled into the program, although it will be ignored if `#MEMORY` is not set.

Compilation of assertions will be disabled along with other debugging functions if `-g` is passed to Rlc or `__NoDebug__` is defined. You can also disable assertions

only, without affecting other debugging code, by passing the `--no-assert` option to Rlc or defining the symbol `__NoAssert__`. Note, however, that any assertions which can be checked at compile-time will *always* be checked, regardless of these settings.

__DebugMessage(*message*)

Prints *message* to the debug console (open with F8).

__DebugMsgBox(*message*, *[type]*): store

Displays a message box containing *message*, which can be an arbitrary string or integer expression.

type can have the following values:

DEBUG_OK	'OK' button only. This is the default if no type is given.
DEBUG_OKCANCEL	'OK' and 'Cancel' buttons. Returns 0 for 'OK' and 1 for 'Cancel'.
DEBUG_YESNO	'Yes' and 'No' buttons. Returns 0 for 'Yes' and 1 for 'No'.

__DebugInputInt(*prompt*, *[default]*, *[min*, *max]*): integer

Displays an input box requesting an integer value. If *default* is given, it sets the default value. If *min* and *max* are given, the return value will be constrained such that it falls within the bounds they define.

__DebugInputStr(*prompt*, *[default]*): string

Displays an input box requesting a string value. If *default* is given, it sets the default value, otherwise the default is the empty string.

__DebugStartTimer(*[index]*)
__DebugGetElapsed(*[index]*): store

Debugging timer controls: start a timer ticking or read its current value. *index*, if given, is an integer from 0 to 31 identifying the debug timer to use.

__SaveBuffer(*file*, *[dc]*)

Saves the contents of *dc* (or the screen buffer if *dc* is not given) to *file*.bmp.

__SaveBufferIdx(*prefix*, *[dc]*)

Saves the contents of *dc* (or the screen buffer if *dc* is not given) to *prefix*[i].bmp, where [i] is a four-digit decimal integer incremented with each call.

Chapter 7

RLdev extension libraries

The RealLive system is missing some useful functionality. RLdev provides various additional features in the form of extension libraries. These libraries are not enabled by default (you must load them explicitly, generally with the `#load` directive, if you wish to use their features), and they are likely always to be more or less experimental.

Unlike the main API, much of the functionality documented in this section is not considered integral to the compiler: the non-integrated parts of these libraries are licensed separately under the terms of the GNU LGPL (see appendix C), plus a special exception, which reads as follows:

As a special exception to the GNU Lesser General Public License, you may include publicly distributed versions of the libraries in a "work that uses the Library" to produce a scenario file containing portions of the libraries, and distribute that scenario file under terms of your choice, without any of the additional requirements listed in clause 6 of the GNU Lesser General Public License. "Publicly distributed versions of the libraries" means either the unmodified libraries as distributed by Haeleth, or modified versions of the libraries that are distributed under the conditions defined in clause 2 of the GNU Lesser General Public License. Note that this exception does not invalidate any other reasons why the scenario file might be covered by the GNU Lesser General Public License.

The effect of this is that you do not need to worry about licensing issues so long as you do not modify the libraries; only if you modify the libraries yourself will you have to distribute any source code or otherwise make any extra effort to comply with the terms of the LGPL.

7.1 rlBabel: a flexible rendering engine for international text

RealLive does not natively support non-Japanese text. The rlBabel library is provided as a solution to this issue.

To use it, you must write your text in UTF-8 and compile it with an output transformation (as described in 4.5.2). You must not disable RLdev metadata when compiling: rlBabel reads this metadata at runtime to determine how to display text in a given scenario file. If no metadata is present, it is assumed that the text in the file is Japanese. (This can be useful; rlBabel makes it possible to display Japanese text on

any platform, even those such as English Windows 98 which are otherwise incapable of it.)

It follows that you can safely mix multiple character sets in a game, provided only one is used in any given scenario; and you can safely mix translated and untranslated text in a translation, provided that all included scenarios are completely translated.

You must also ensure that a rlBabel DLL is accessible at runtime. This latter requirement is fulfilled differently depending on the interpreter you are using:

1.2 to 1.2.4
The DLL to use is `rtl/rlBabelF.dll`. Copy this into your game directory, and modify the game's initialisation code to include the line

LoadDLL(0, `'rlBabelF'`)

The initialisation issues described under 5.6 can be ignored in the case of rlBabel, which artificially bumps its own reference count to prevent the essential code being unloaded. Provided the DLL is loaded once at system startup, you do not have to worry about reloading it after resets.

1.2.5+
The DLL to use is `rtl/rlBabel.dll`. Copy this into your game directory, and modify `gameexe.ini` to include a line along the lines of

#DLL.000 = `"rlBabel"`

where the 000 should be increased accordingly if other DLLs are already being used.

Since the codeset detection code requires access to details of the internal interpreter state which are not directly exposed to extension DLLs, it is also necessary to provide rlBabel with a *map file* which identifies the memory addresses to query. Ready-defined map files for about 20 different interpreters are supplied in the `rtl/` directory: for example, if your game runs in RealLive 1.3.1.0, you need to copy the file `rtl/1.3.1.0.map` to the game directory for dynamic formatting to work.

If you are using an interpreter for which no map is available, you will have to generate one. There is a program called "rlBabel-GenMap" which can do this automatically: it should be available from the same sources as RLdev.

Note that map files are not necessary for the 1.2.3.5 interpreter supplied with *Clannad* or for the 1.2.6.8 interpreter supplied with *Kanon*, as the mappings for these two versions are built into the DLL.

7.1.1 Configuring fonts and international names

RealLive will normally select a Japanese font, and interpret the default names (defined with `#NAME` and `#LOCALNAME` in `gameexe.ini`) as being in the default Japanese encoding. Neither of these is likely to be a desirable effect if you are using a different codepage. To solve these issues, rlBabel introduces some extensions to `gameexe.ini`.

The extensions to the font setting system (which are generally useful) are described in 8.2.2.

To provide default names in a non-Japanese encoding, simply enter the names you wish to use as text in the normal Windows encoding for that codepage, and add one additional variable:

#NAME_ENC = *enc*

enc indicates the encoding used: 1 is Chinese, 2 is Western, and 3 is Korean.

7.1.2 Dynamic lineation and proportional text

This functionality is currently experimental; remember to test your code very carefully if you use it.

Chinese and Japanese text require very little in the way of lineation. Western text is different. Since RealLive is Japanese in origin, it provides very little support for lineation. This has been a consistent problem for those attempting to run English games on the platform.

Extension DLLs (see 5.6) provide sufficient functionality for this to be worked round. C++ code in a DLL can easily take a string, tokenise it, and feed back tokens to RealLive for display. This functionality is provided in rlBabel. In addition to this, the rlBabel system can be used to enable use of Western characters and punctuation not present in Japanese, and by taking advantage of direct access to the Windows API it is possible to query character widths at runtime and so render text proportionally.

Using dynamic formatting

At present, your code must use the `western` character set and output transformation to use the dynamic lineation and proportional text features.

To enable the feature, add the line

```
#load'rlBabel'
```

to your global header (or to the start of every file that displays text). This enables the compilation mode that generates code for rlBabel. You must also be certain to call `rlcInit()` when the interpreter first loads. (This automatically calls `LoadDLL()` in versions of RealLive where this is required, so you need not do that yourself.)

The lineation code requires a handful of global variables; if you are using `rlcSetAllocation()` to modify the default allocation settings (which is recommended, and will be absolutely necessary if you are modifying an existing program), you must do so either in a global header or at the top of *every* source file that allocates variables, and in all cases *before* loading the library.

Using hypertext with dynamic formatting

When dynamic formatting is enabled, it is also possible to use hypertext with the `\g` control code. There is some overhead involved in hypertext processing, so this is disabled by default: you must enable it explicitly by adding the line

```
#define __EnableGlosses__
```

before you load the rlBabel library.

When a hyperlink is clicked on, rlBabel fills a string variable with the text associated with the link and passes it to a handler. By default, this simply displays the text in a standard Windows message box. You can override this behaviour by defining a custom handler. The syntax for this is

```
#inline rlBabelGlossCallback (gloss_var):
  // your code
  ;
```

where `gloss_var` is bound to the string variable containing the text. This should also be defined before the rlBabel library is loaded.

Additional features provided in the dynamic formatting library

Text can be italicised by using the new `\b` and `\u` control codes (not to be mistaken for "bold" and "underline"!):

`\b`

Begin an emphasis block. The text that follows will be italicised, up to the next `\u` code or the end of the current string.

`\u`

End an emphasis block.

For example, `'R2D2! It \bis\u you!'` outputs "R2D2! It *is* you!".

Limitations of dynamic formatting

Certain RealLive features are not available in rlBabel-formatted text. In particular, the `\ruby` control code is not supported at present.

Proportional text formatting has not been extended to cover all cases. In particular, text displayed by `select_s()`, `objOfText()`, `grpTextout()`, and related functions still uses the fixed advance width system built into RealLive. Support for these functions is planned for a future release.

Using bitmapped characters in text

When using proportional text output, bitmapped characters printed with the `\e` and `\em` codes are currently not permitted in `select()` calls. They can be used as normal in regular text.

By default, bitmapped characters are exactly one em wide. When using proportional text output, it is possible to specify widths for individual characters by defining a new variable in `gameexe.ini`:

#RLBABEL_E_MOJI_WIDTHS = *width*[, *width*...]

Define widths for bitmapped characters. Each *width* is a percentage of an em square for the corresponding character: that is, the first *width* defines the width of character 0, the second defines character 1, and so on. Any widths left unspecified default to 100, i.e. the normal width.

Utility functions

The rlBabel DLL implements a number of additional functions which may be of general utility, but which are not provided directly by RealLive itself (or, to be precise, are not *known* to be provided by RealLive).

These functions are defined in the `rlBabel` header file. If you wish to use them in a program which does not actually use rlBabel for dynamic lineation of Western text, you will need to disable dynamic lineation manually by adding the line

```
#set __DynamicLineation__ = 0
```

immediately after you load the header.

MessageBox(*expr*, *[title]*): store

Pops up a message box displaying the result of the given expression. Returns 1 if the operation succeeded, 0 otherwise.

Unlike `___DebugMsgBox()`, this function is not ignored outside debug mode, and the title of the message box can be customised.

GetTextWindow(): store

Returns the index of the currently active text window.

SetNameMod(*value*, *[window]*): store

Modifies the value of `#WINDOW.NAME_MOD`, for *window* if given, or else for the currently active window. Returns the previous value.

GetNameMod(*[window]*): store

Returns the current value of `#WINDOW.NAME_MOD`, for *window* if given, or else for the currently active window.

SetCurrentWindowName(*name*): store

Sets the speaker name associated with the current window, without displaying any text. That is to say,

```
SetCurrentWindowName('speaker')
'speaker "Examples are easier to think of when it\'s not so late."'
```

is usually roughly equivalent to

```
'\{speaker} "Examples are easier to think of when it\'s not so late."'
```

The return value is always 1 if no error occurred. (If an error occurs, it is unlikely that this function will return at all, but hypothetically it might conceivably return 0 if some deity intervened.)

7.2 Textout: a compatible rendering engine for Western text

The Textout library is currently experimental; remember to test your code very carefully if you use it.

The rlBabel library cannot be used by RealLive versions below 1.2, or by RealLive 1.2 to 1.2.5 where an extension DLL is already in use, or by games which for whatever reason need to display Japanese text as well as a Western language. To get round this issue, an alternative library is provided which implements a similar dynamic lineation engine in RealLive bytecode itself. When the Textout library is loaded, text strings are tokenised at compile-time, and fed at runtime into a rendering loop that

keeps a record of the state of the various text output windows and inserts line breaks automatically as appropriate.

Compared to rlBabel, the Textout library has a number of limitations. First and foremost, it does not alter any RealLive behaviours directly: so players will be limited to Japanese fonts and characters once more, and only monospaced text will be displayed. It is also slower, and causes an even greater increase in the size of the compiled code. For these reasons, rlBabel is usually a better choice if you are able to use a supported interpreter.

To use this feature, insert the line

```
#load'textout'
```

at the start of each file in the project, or in a project header, and ensure that the `rlcInit()` function is called when the program begins. The Textout library reserves a fair amount of memory, depending on how many text windows are defined; if you are using `rlcSetAllocation()` to modify the default allocation settings (which is recommended, and will be absolutely necessary if you are modifying an existing program), you must do so *before* loading the library.

Usage once the library is loaded is automatic. It can be disabled by setting the value of the constant __DynamicLineation__ to 0, but this is not recommended.

Additional diagnostics can be enabled by defining the symbol __DebugTextout__ before loading the library.

Note that while the Textout library compiles to valid RealLive bytecode, it is formed such that it cannot be disassembled to valid Kepago again. If you disassemble a file that uses it, you will *not* be able to recompile the disassembled source code. (You may consider this an advantage, if you do not want your program to be easily modifiable by unauthorised third parties!)

The actual rules governing RealLive's text windows are complex and filled with special cases. This library currently uses a simplified model which does not always perfectly match the true behaviour. In particular, RealLive's special handling of certain double-byte characters, such as punctuation and small kana, is *not* taken into account, which may well lead to incorrect lineation of strings containing these characters. The safest option is not to mix western and Japanese text in one window.

Chapter 8

Gameexe.ini reference

This incomplete chapter will eventually document the file `gameexe.ini`. At the moment it only covers things I've noted down as I work, in no particular order.

Note that `gameexe.ini` is case-sensitive.

8.1 Interpreter configuration

8.1.1 Main window settings

#CAPTION = "*title*"

Sets the caption of the window to *title*.

#SUBTITLE = *flag*

Set to enable display of game titles defined with `title()` calls in the title bar of the game window (after the string defined with `#CAPTION`).

#SCREENSIZE_MOD = *flag*

Game resolution: 0 is 640 × 480, 1 is 800 × 600.

#INIT_SCREENMODE = *flag*

The game defaults to windowed mode if *flag* is 1, full-screen mode otherwise.

8.1.2 Locations

#REGNAME = "*key*"

Game configuration data will be sought in the registry at the location

HKEY_CURRENT_USER\Software*key*

The data stored in the registry are those related to the particular installation of the game, such as the path of the installation disc, the parts of the data which have been installed, and the location in which saved games should be stored.

xclannad uses *key* to identify the game for various other purposes, such as in the construction of save filenames.

#FOLDNAME.*ext* = "*folder*" = *archive* : "*arcname*"

Configures the interpreter to look for data files with the extension *ext* in the named *folder*. If *archive* is 1, the archive *arcname* will also be searched.

Known extensions:

ANM	PDT animations
ARD	Area definitions
BGM	Music files (*.nwa, *.ogg, etc.)
DAT	General data files
G00	Graphics
GAN	G00 animations
HIK	
KOE	Voice data
M00	
MOV	Video files (*.avi, *.mpg, etc.)
PDT	PDT graphics (used for cursors)
TXT	Bytecode
WAV	Sound effects

As a minimum, all projects appear to contain the line

#FOLDNAME.TXT = "DAT" = 1 : "SEEN.TXT"

#CGTABLE_FILENAME = "*file*"

Sets the name of the data file used to associate graphics and sections of text with special events, so the interpreter can keep track of which have been viewed. This is normally mode.cgm.

#KOESYNC_FILENAME = "*file*"

Where *file* is set, it is normally koeanm.dat. Its purpose is unclear, but from the name one assumes it has something to do with the synchronisation of voices and animations.

#TONECURVE_FILENAME = "*file*"

Where *file* is set, it is normally tcdata.tcc. Its purpose is unclear.

#BUSTSHOT_FILENAME = "*file*"

Where *file* is set, it is normally bustshot.bst. The mind boggles at its possible uses.

#MANUAL_PATH = "*file*"

Sets the path relative to gameexe.ini of a file, typically the game's manual, which is invoked (as though with the shell() function) when the player selects syscom 24 "Open manual" from the system command menu.

This variable does not exist in RealLive builds prior to 1.2.6.

8.1.3 Bytecode entrypoints

#SEEN_START = *index*

Defines the program entrypoint. Execution will begin at SEEN*index* entrypoint 0 whenever the interpreter is started or reset.

#SEEN_MENU = *index*

Defines the menu entrypoint. Execution will pass to SEEN*index* entrypoint 0 whenever the interpreter is returned to the title menu, either by a call to the ReturnMenu() or MenuReturn() functions, or by selecting syscom 28 "Return to menu" from the built-in system command menu.

#CANCELCALL = *scenario*, *entrypoint*
#CANCELCALL_MOD = *flag*

It is possible, using these variables, to install a custom menu system in place of the built-in system command menu described in 5.5.

If #CANCELCALL_MOD is set to zero, the built-in menu is used. If it is non-zero, the #CANCELCALL settings are used, and *scenario* will be called at the given *entrypoint* whenever the player invokes the menu; your custom menu routine should call rtlCancel() to return control to the point at which the menu was invoked.

8.1.4 Memory settings

#*intvar*[*index*] = *value*
#*strvar*[*index*] = "*value*"

Sets the initial value of *intvar*[*index*] or *strvar*[*index*]. Where the variable space is local, it will be initialised to *value* each time the interpreter is started; where it is global, it will be initialised to *value* the first time the interpreter is started, but thereafter it will retain its value as normal. (Refer to 5.4 for details of local and global memory spaces.)

Valid values of *intvar* are all valid integer addresses in all addressing modes, capitalised and without the int prefix (e.g. #C[100], #G2B[4000]). Valid values of *strvar* are S and M.

#NAME.*var* = "*value*"
#LOCALNAME.*var* = "*value*"

Sets the initial value of the name variable *var* to *value*. #NAME selects the global name variable space, and #LOCALNAME the local space. The situations in which these settings are used are identical to those described above for regular variables. (Refer to 5.4.2 for details of name variables.)

Valid values of *var* are all alphabetical name variable identifiers: that is, A to Z and AA to ZZ.

#NAME_MAXLEN = *len*

Sets the maximum length of a name variable to $len \times 2$ bytes (i.e. *len* dual-byte characters). The length is always between 12 and 20 bytes; values of *len* less than 6 or greater than 10 are treated as 6 and 10 respectively.

8.2 Text output

Message window configuraton is quite complicated, and not well understood. The incomplete documentation here contains some details; if in doubt, you'd be better off examining what's done in actual games and copying details from there...

8.2.1 General configuration

#WINDOW_MOVE_MODE = *mode*

If window movement is enabled, controls what is rendered while a window is being dragged:

- **0** display only a box outline
- **1** render border only
- **2** render border and background
- **3** render border, background, and text

#WINDOW_ATTR = *r*, *g*, *b*, *alpha*, *filter*

Defines the default window attributes, as read by `DefWindowAttr()`. These set the the RGB components of a window's background colour, the *alpha* value with which it is composited with the image behind, and a value *filter* which determines the composition mode: if it is 0, a subtractive filter is used, while if it is 1, a straightforward alpha filter is used.

These settings can be overridden for individual windows by using `#WINDOW.ATTR_MOD` to enable the use of that window's `#WINDOW.ATTR` instead.

8.2.2 Global font settings

#INIT_FONT_TYPE = *type*

Selects a basic font style to use by default. If *type* is 0, the game will use MS Gothic; if it is 1, the game will use MS Mincho. If the selected font is not available on the player's system (or if the player is using a non-Japanese codepage), they will be forced to select a custom font the first time text is displayed.

As an RLdev extension, if the RLdev Unicode mode has been enabled by loading the rlBabel DLL (see 7.1), you can specify a different default font by setting *type* to 2 and using `#INIT_FONT_NAME`. If rlBabel is not loaded, this will have no effect.

#INIT_FONT_NAME = "*font*"

This is an RLdev extension. It is only available when the rlBabel DLL is loaded (see 7.1); otherwise it has no effect.

If `#INIT_FONT_TYPE` is 2, then *font* is used as the default font for the game.

#INIT_FONT_QUALITY = *quality*

Sets the default font quality to use. The values of *quality*, and their meanings, are the same as for `SetFontQuality()`.

#INIT_FONT_WEIGHT = *weight*

Sets the default font weight to use. The values of *weight*, and their meanings, are the same as for SetFontWeight().

8.2.3 Defining a window

Up to 64 windows may be used. Each is defined by a WINDOW block in gameexe.ini. Each line of a WINDOW block is of the form #WINDOW.*index*.*variable* = ..., which defines the setting of *variable* for window *index*. Possible *variable*s are described in this section.

8.2.3.1 Appearance

#WINDOW.*index*.ATTR_MOD = *mod*
#WINDOW.*index*.ATTR = *r*, *g*, *b*, *alpha*, *filter*

If *mod* is 0, the window uses the default attributes defined by #WINDOW_ATTR; if *mod* is 1, the local settings defined by *r*, *g*, *b*, *alpha*, and *filter* are used instead for this window.

#WINDOW.*index*.POS = *origin* : *x*, *y*

Determines the initial position of the window. *x* and *y* are the absolute distance between an edge of the screen, determined by *origin*, and the nearest edge of the window. Values of *origin* are:

0 Top and left
1 Top and right
2 Bottom and left
3 Bottom and right

#WINDOW.*index*.WAKU_SETNO = *waku*

Define the window style. *waku* is the number of a waku set as defined with a #WAKU block.

#WINDOW.*index*.WAKU_MOD = *waku_mod*
#WINDOW.*index*.WAKU_NO = *pattern*

If *waku_mod* is zero (the default), the window respects the global border pattern settings as defined with SetWakuAll(), and uses the corresponding pattern from its associated waku set. If it is non-zero, then the window always uses *pattern*.

#WINDOW.*index*.KEYCUR_MOD = *type* : *x*, *y*

Determines the position of the keycursor (the animated cursor that appears when the game is waiting for a click to move to the next page of text).

If *type* is 0, the cursor appears at the bottom right corner of the text area; if it is 1, it appears directly after the final character printed; if it is 2, it appears at (*x*, *y*) relative to the top left of the text area. (*x* and *y* are ignored when *type* is 0 or 1.)

8.2.3.2 Animations

#WINDOW.*index*.OPEN_ANM_MOD = *mod*
#WINDOW.*index*.CLOSE_ANM_MOD = *mod*

Define the animation type to use for opening or closing the window. Possible values of *mod* in each case are:

0	Instantaneous
1	Fade
2	Scroll off/on via top of screen
3	Scroll off/on via bottom of screen
4	Scroll off/on via left of screen
5	Scroll off/on via right of screen
6	Scroll off/on vertically via nearest edge of screen
7	Scroll off/on horizontally via nearest edge of screen
8	Scroll off/on via nearest edge of screen (?)
9	Expand horizontally and vertically
10	Expand horizontally, contract vertically
11	Expand horizontally
12	Contract horizontally, expand vertically
13	Contract horizontally and vertically
14	Contract horizontally
15	Expand vertically
16	Contract vertically

These values can be queried and modified at runtime with functions such as `GetOpenAnmMod()`.

#WINDOW.*index*.OPEN_ANM_TIME = *ms*
#WINDOW.*index*.CLOSE_ANM_TIME = *ms*

If the open or close animation mode is anything other than 0, the times given here determine how long the animations take to run, in milliseconds. These values can be queried and modified at runtime with functions such as `GetOpenAnmTime()`.

8.2.3.3 Text settings

#WINDOW.*index*.MOJI_CNT = *horizontally*, *vertically*

Sets the size of the text area within the window. It will be set to hold exactly this number of characters at the default font size.

#WINDOW.*index*.MOJI_POS = *top*, *bottom*, *left*, *right*

Determines the amount of padding to insert at each edge of the text area, in pixels. The first character on each screen will be printed at (*top*, *left*).

#WINDOW.*index*.MOJI_SIZE = *size*

Sets the default size for text in the window, in pixels. This is the height of a character, and (since text is always fixed-width) the width of a zenkaku character, or twice the width of a hankaku character.

#WINDOW.*index*.MOJI_REP = *x*, *[y]*

Adds or removes spacing between characters (*x*) and lines (*y*). Values of 0 leave the spacing as that determined by `#WINDOW.MOJI_SIZE`; positive values expand spacing, and negative values condense it. These values are measured in pixels, and they are absolute: they do not scale with the text size.

#WINDOW.*index*.LUBY_SIZE = *size*

Sets the *size*, in pixels, of text displayed with the `\ruby` control code. If *size* is 0, ruby text is disabled for this window.

Space is always left for ruby text, regardless of whether it is present or not: thus the height of each line in a window is defined as $MOJI\_SIZE + MOJI\_REP_y +$

LUBY_SIZE.

#WINDOW.*index*.INDENT_USE = *flag*

This feature is not available in RealLive prior to 1.2.

This flag affects what happens when reported speech is longer than one line. If *flag* is 1, then all subsequent lines are aligned with the opening quotation mark; if *flag* is 0, then no indentation is performed, and subsequent lines begin at the left margin instead.

#WINDOW.*index*.MOJI_SHADOW = *flag*

Determines whether text within this window is rendered with a shadow or not.

#WINDOW.*index*.KINSOKU_USE = *flag*

Determines whether to use Japanese line-breaking rules or not. If *flag* is set, RealLive will avoid breaking lines before certain punctuation marks.

#WINDOW.*index*.R_COMMAND_MOD = *flag*

Determines how the window will react to `pause()` calls. If *flag* is 0, then `pause()` behaves identically to `page()`, i.e. the active text window is cleared when the game continues, and all other open text windows are closed. If *flag* is 1, `pause()` ends a paragraph rather than a page, inserts a line break instead of clearing the window, and leaves other text windows unmodified.

8.2.3.4 Select window settings

The `select_w()` function makes use of regular text windows, but its treatment of them differs somewhat from that of the usual text display routines. In particular, it resizes the text area dynamically based on the options it contains. This section documents settings that take effect only when a window is used with `select_w()`.

#WINDOW.*index*.SELCOM_USE = *flag*

If *flag* is 1, the window can be used by `select_w()`. If it is 0, then calls to `select_w()` which reference the window will fail and return the error value −2.

#WINDOW.*index*.MOJI_MIN = *width*, *height*

Determine the size of the text area. It will be dynamically sized to fit the number of options presented and the width of the widest option, but it will never be smaller than the size given here.

#WINDOW.*index*.SELCOM_SETPOS = *horizontal*, *vertical*, *x*, *y*

The values of *horizontal* and *vertical* control where the window is placed when used by a `select_w()` call:

0	use settings from `#WINDOW.POS`
1	centered
2	relative to left or top of screen
3	relative to right or bottom of screen

x and *y* are used only in the latter two cases, where they specify the spacing to use between the window and the relevant edge of the screen.

For example, values $0, 0, n, n$ will cause the window to be located in exactly the same place as it would be used for regular text display, while values $1, 2, n, 100$ will cause it to be centered horizontally and positioned 100 pixels from the top of the screen. (n here stands for any number, since these values are not used.)

#WINDOW.*index*.SELCOM_MOJIPOS = *flag*

If *flag* is zero, items are left aligned; if it is nonzero, items are centered.

#WINDOW.*index*.SELCOM_CURSORSELECT = *flag*
#WINDOW.*index*.SELCOM_CURSORNO = *cursor*
#WINDOW.*index*.SELCOM_MOJIDARK = *amount*

If *flag* is 0 or −1, the selected item is highlighted by inverting its foreground and background colours, and *cursor* and *amount* are ignored.

If *flag* is 1, the selected item has the cursor #CURSOR.*cursor* displayed beside it, and *un*selected text is darkened by *amount*. *amount* is a value from 0 (no effect) to 255 (unselected text is black); a typical value is 64.

#WINDOW.*index*.SELCOM_MOUSESET = *flag*

If *flag* is non-zero, the mouse cursor will automatically be moved to the first option.

8.2.3.5 Name window settings

#WINDOW.*index*.NAME_MOD = *name_mod*

By default, character names (as identified with the \name control code or its abbreviation \{}) are displayed inline in the text window. This is not the only option. Valid settings of *name_mod* are:

0 Display names inline (default)
1 Display names in a separate window.
2 Do not display names.
3 Same as 2?

The other variables in this section are only used when *name_mod* is 1.

#WINDOW.*index*.NAME_POS = *x*, *y*

By default, the name window is positioned above the text window and aligned with its left edge. The values specified here for *x* and *y* can be used to adjust this position.

#WINDOW.*index*.NAME_MOJI_SIZE = *size*

Sets the size of text in the name window, in pixels.

#WINDOW.*index*.NAME_MOJI_REP = *x*

Adjusts the character spacing in the name window by *x* pixels.

#WINDOW.*index*.NAME_MOJI_POS = *x*, *y*

Sets the spacing around the name. This is applied on all sides: for example, if *y* is 50, then the name will be centered vertically in a text area $100+NAME\_MOJI\_SIZE$ pixels high.

#WINDOW.*index*.NAME_MOJI_MIN = *width*

The name window will expand and shrink to fit the name provided, but it will never be fewer than *width* zenkaku characters wide.

#WINDOW.*index*.NAME_WAKU_DIR = *flag*

Not fully understood.It appears that setting *flag* to a non-zero value causes the horizontal component of #WINDOW.NAME_POS to be interpreted relative to the left edge of the screen, rather than the left edge of the text area of the window. Further investigation is needed.

#WINDOW.*index*.NAME_WAKU_SETNO = *waku*

Defines the window style to use for the name window. *waku* is the number of a waku set as defined with a #WAKU block. It need not be the same style as used by the main window.

#WINDOW.*index*.NAME_CENTERING = *flag*

If *flag* is zero, the name will be aligned with the left of the name window; if *flag* is non-zero, it will be centered horizontally.

8.2.3.6 Local functionality toggles

#WINDOW.*index*.EXBTN_nnn_USE = *flag*

Enable or disable specific custom window buttons for the window. *nnn* is a value from 0 to 7.

To use each custom button, in addition to enabling this flag you must also enable #WINDOW_EXBTN_USE, define #WBCALL.*nnn* and create the procedure it calls, and define a #WAKU.EXBTN_nnn_BOX for the button in the window style(s) used by the window.

#WINDOW.*index*.MOVE_USE = *flag*

If the global #WINDOW_MOVE_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.CLEAR_USE = *flag*

If the global #WINDOW_CLEAR_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.READJUMP_USE = *flag*

If the global #WINDOW_READJUMP_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.AUTOMODE_USE = *flag*

If the global #WINDOW_AUTOMODE_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.MSGBK_USE = *flag*

If the global #WINDOW_MSGBK_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.MSGBKLEFT_USE = *flag*

If the global #WINDOW_MSGBKLEFT_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.MSGBKRIGHT_USE = *flag*

If the global #WINDOW_MSGBKRIGHT_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

#WINDOW.*index*.KOEPLAY_USE = *flag*

This variable is not available in RealLive prior to 1.3.4.

If the global #WINDOW_KOEPLAY_USE flag is set, this flag is used to decide on a window-by-window basis which windows it applies to.

8.2.3.7 Miscellaneous settings

flag/

If *flag* is 1, text displayed in the window will be stored in the message backlog; if it is 0, the window's contents will be ignored when reviewing previously-read text.

#WINDOW.*index*.MESSAGE_MOD = *mod*
#WINDOW.*index*.NOVELBACK = *x1*, *y1*, *x2*, *y2*

The purpose and use of these variables is not understood. *mod* is usually 0, and (x_1, y_1), (x_2, y_2) generally describe the visible area of the screen.

#WINDOW.*index*.FACE.*n* = *x*, *y*, *behind*, <?>, <?>

Define a face slot for the window. These are settings used when displaying character portraits with the `FaceOpen()` function. Up to eight slots can be defined for each window; one `FACE` line is used to define each, with the first being `FACE.000`.

x and *y* are the position at which character portrait *n* will be displayed, relative to the top left of the text area as defined by `#WINDOW.MOJI_POS`. *behind* determines whether the portrait will be layered in front of (0) or behind (1) the window background. The meaning of the other two values is unknown.

8.3 Audio

#CDTRACK = *from* – *to* – *loop* = "*name*"

Defines track *name* to be CD audio. *from*, *to*, and *loop* define the start, end, and position to return to when looping respectively; each is of the form *track*:*m*:*s*:*ms*, where *track* is a track on the CD, and *m*:*s*.*ms* defines a position in minutes, seconds, and milliseconds.

name is an arbitrary string, which is passed to music playback functions (described in 6.9.2) to operate on this track.

#DSTRACK = *from* – *to* – *loop* = "*file*" = "*name*"

Defines track *name* to be DirectSound audio, usually in the VisualArt's `nwa` format.

The track is taken from *file*; *from*, *to*, and *loop* define positions in that file, as in `#CDTRACK`, but in this case each is a single integer. (I'm not sure what the unit is.)

name has the same semantics as in `#CDTRACK`; it need not be related to *file* in any way. DirectSound music is played using the same functions as CD audio.

#SE.*index* = "*file*" = *channel*

Associates interface sound *index* with *file*.wav. *index* is an integer; values of up to 20 are known to be valid. *channel* is the interface sound channel to use for this sound effect, and can be from 0 to 15. You must place sounds that will be played at the same time in different channels, but there seems to be no other limitation.

The functions which operate on interface sounds are described in 6.9.4.

8.4 Debugging

#MEMORY = *flag*

Set to enter debug mode. This unlocks all the various debug menus and data readouts, and enables all the debugging functions described in 6.15.

#DEBUG_MESSAGE_LOG = *flag*

If set, all debug messages will be written to a log file. (They can be viewed regardless by opening the "debug messages" window.)

Undocumented variables

Some variables are not yet documented. If you followed a link here, you just found a reference to one in the documentation. Sorry.

#SYSCOM_MOD = *flag*

If set, configuration options are placed in a submenu; otherwise the menu is flat.

#SYSCOM_MOD2 = *flag*

If set, popup dialogs are used to confirm actions like exiting the game, instead of the default submenus.

#SAVELOADDLG_USE = *flag*

If set, popup dialogs are used to select saved game slots, rather than the default submenus.

#DLL.*idx* = "*filename*"

This variable is not available in RealLive prior to 1.2.5.

Loads *filename* into extension DLL slot *idx*. The `.dll` extension can be omitted. See 5.6 for details of the extension DLL system.

Appendix A

VisualArt's–RLdev name equivalences

This appendix gives a list of known VisualArt's function names, and their RLdev equivalents in the Kepago/RealLive API (where known). The VisualArt's names, on the left, are the ones that will be visible when debugging.

BANKALLOC	allocDC()
BANKBANK	
BANKFREE	freeDC()
BG	grpOpenBg()
BGCHR	grpMulti()
BGCHRDC	grpMulti()
BGCHRKEEP	
BOXANTIMASKCOPY	grpCopyInvMask()
BOXANTIMASKCOPYMASK	grpMaskCopyInvMask()
BOXANTIMASKCOPYMASK_ADD	grpMaskAddInvMask()
BOXANTIMASKCOPYMASK_SUB	grpMaskSubInvMask()
BOXANTIMASKCOPY_ADD	grpAddInvMask()
BOXANTIMASKCOPY_SUB	grpSubInvMask()
BOXANTIMASKDARK	
BOXANTIMASKEMPTY	
BOXANTIMASKFILL	
BOXANTIMASKMONO	
BOXANTIMASKNEGA	
BOXANTIMASKRGB	
BOXBG	
BOXBGCHR	
BOXBGCHRDC	
BOXBGCHRKEEP	
BOXCHR	
BOXCOPY	grpCopy()
BOXCOPYAND	grpAnd()
BOXCOPYEASY	grpCMaskCopy()
BOXCOPYMASK	grpMaskCopy()

BOXCOPYMASKBLEND	grpMaskBlend()
BOXCOPYMASK_ADD	grpMaskAdd()
BOXCOPYMASK_SUB	grpMaskSub()
BOXCOPYOR	grpOr()
BOXCOPY_ADD	grpAdd()
BOXCOPY_SUB	grpSub()
BOXDARK	grpLight()
BOXEMPTY	grpOutline()
BOXFADE	grpFade()
BOXFILL	grpFill()
BOXGRP	
BOXMASKCOPY	grpCopyWithMask()
BOXMASKCOPYMASK	grpMaskCopyWithMask()
BOXMASKCOPYMASK_ADD	grpMaskAddWithMask()
BOXMASKCOPYMASK_SUB	grpMaskSubWithMask()
BOXMASKCOPY_ADD	grpAddWithMask()
BOXMASKCOPY_SUB	grpSubWithMask()
BOXMASKDARK	
BOXMASKEMPTY	
BOXMASKFILL	
BOXMASKMONO	
BOXMASKNEGA	
BOXMASKRGB	
BOXMONO	grpMono()
BOXNEGA	grpInvert()
BOXNUMBER	grpNumber()
BOXNUMBERMASK	grpMaskNumber()
BOXPDTEXPAND	grpLoad()
BOXPDTEXPANDCHR	grpMaskLoad()
BOXRGB	grpColour()
BOXROTATECOPY	grpRotate()
BOXROTATECOPYMASK	grpMaskRotate()
BOXROTATECOPYMASK_ADD	grpMaskRotateAdd()
BOXROTATECOPYMASK_SUB	grpMaskRotateSub()
BOXROTATECOPY_ADD	grpRotateAdd()
BOXROTATECOPY_SUB	grpRotateSub()
BOXSCROLL	grpPan()
BOXSHIFT	grpShift()
BOXSLIDE	grpSlide()
BOXSTRETCH	grpMaskStretchBlt()
BOXSTRETCH	grpStretchBlt()
BOXSTRETCHMASK	
BOXSTRETCHMOVE	grpZoom()
BOXWAIP	
BOXXCHG	grpSwap()
CHR	grpMaskOpen()
FARRETURN	rtl()
GCLS	wipe()

GRP	grpOpen()
GRPKEEP	
INIT_CALLSTACK	CallStackClear()
KANJI	grpTextout()
MASKBANKCOPY	
NOP	stackNop()
PDTEXPAND	grpBuffer()
PDTEXPANDCHR	grpMaskBuffer()
PDTEXPANDMASK	grpLoadMask()
PIXELPUT	
PIXELSET	
RECTANTIMASKCOPY	recCopyInvMask()
RECTANTIMASKCOPYMASK	recMaskCopyInvMask()
RECTANTIMASKCOPYMASK_ADD	recMaskAddInvMask()
RECTANTIMASKCOPYMASK_SUB	recMaskSubInvMask()
RECTANTIMASKCOPY_ADD	recAddInvMask()
RECTANTIMASKCOPY_SUB	recSubInvMask()
RECTANTIMASKDARK	
RECTANTIMASKEMPTY	
RECTANTIMASKFILL	
RECTANTIMASKMONO	
RECTANTIMASKNEGA	
RECTANTIMASKRGB	
RECTBG	
RECTBGCHR	recMulti()
RECTBGCHRDC	recMulti()
RECTBGCHRKEEP	
RECTCHR	
RECTCOPY	recCopy()
RECTCOPYAND	recAnd()
RECTCOPYEASY	recCMaskCopy()
RECTCOPYMASK	recMaskCopy()
RECTCOPYMASKBLEND	recMaskBlend()
RECTCOPYMASK_ADD	recMaskAdd()
RECTCOPYMASK_SUB	recMaskSub()
RECTCOPYOR	recOr()
RECTCOPY_ADD	recAdd()
RECTCOPY_SUB	recSub()
RECTDARK	recLight()
RECTEMPTY	recOutline()
RECTFADE	recFade()
RECTFILL	recFill()
RECTGRP	
RECTMASKCOPY	recCopyWithMask()
RECTMASKCOPYMASK	recMaskCopyWithMask()
RECTMASKCOPYMASK_ADD	recMaskAddWithMask()
RECTMASKCOPYMASK_SUB	recMaskSubWithMask()
RECTMASKCOPY_ADD	recAddWithMask()

RECTMASKCOPY_SUB	recSubWithMask()
RECTMASKDARK	
RECTMASKEMPTY	
RECTMASKFILL	
RECTMASKMONO	
RECTMASKNEGA	
RECTMASKRGB	
RECTMONO	recMono()
RECTNEGA	recInvert()
RECTNUMBER	
RECTNUMBERMASK	
RECTPDTEXPAND	recLoad()
RECTPDTEXPANDCHR	recMaskLoad()
RECTRGB	recColour()
RECTROTATECOPY	recRotate()
RECTROTATECOPYMASK	recMaskRotate()
RECTROTATECOPYMASK_ADD	recMaskRotateAdd()
RECTROTATECOPYMASK_SUB	recMaskRotateSub()
RECTROTATECOPY_ADD	recRotateAdd()
RECTROTATECOPY_SUB	recRotateSub()
RECTSCROLL	recPan()
RECTSHIFT	recShift()
RECTSLIDE	recSlide()
RECTSTRETCH	recMaskStretchBlt()
RECTSTRETCH	recStretchBlt()
RECTSTRETCHMASK	
RECTSTRETCHMOVE	recZoom()
RECTWAIP	
RECTXCHG	recSwap()
RETURN	ret()
SERIALPDTANM	snmPlay(), etc.
SERIALPDTSCROLL	snmScroll(), etc.
SERIALPDTSTRETCHANM	snmStretch(), etc.
SUB_CALLSTACK	CallStackPop()
WAIP	grpDisplay()

Appendix B

The GNU GPL

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307,
USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) <year>  <name of author>
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items — whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
(signature of Ty Coon), 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Appendix C

The GNU LGPL

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages — typically libraries — of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by ob-

taining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

Terms and Conditions For Copying, Distribution and Modification

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) The modified work must itself be a software library.

   (b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   (c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   (d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

   In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that

work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

(a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

(b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

(c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

(d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

(e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   (a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   (b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
```

```
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301,
USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
(signature of Ty Coon), 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

Appendix D

Other licensing information

The compression code used by RLdev is a derivative of Jagarl's LZcomp library, modified slightly to output in the RealLive format. This code bears the following notice:

> Copyright © 2001 Kazunori Ueno(JAGARL) <jagarl@creator.club.ne.jp>
>
> All rights reserved.
>
> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
> 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
>
> THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The code in question can be obtained by extracting the file `src/common/lzcomp.h` from the RLdev source code; at the time of writing, the original file could be found at http://www.creator.club.ne.jp/~jagarl/lzcomp.h.