RLIB Programmers Manual

SICOM Systems, INC.

4140 Skyron Drive, Doylestown PA 18901

Bob Doan

Robert Kratz

Chet Heilman

RLIB Programmers Manual

by **SICOM Systems, INC.**, **4140 Skyron Drive, Doylestown PA 18901**
by Bob Doan

by Robert Kratz

by Chet Heilman

Table of Contents

Chapter 1. Introduction

RLIB in a nutshell

RLIB is a report generation library/language. It takes advantage of today's best web technology including PHP, SQL, and XML. Because speed is an extremely important with web applications, and requests must be completed with sub second response time, c was chosen as the language in which to implement RLIB.

One of the main advantages to RLIB is that you don't have to be a programmer in order to use it. The file format for describing reports is XML. RLIB supports full expression evaluation in human readable format so it is easy to follow the logic behind a report.

This manual assumes you have prior knowledge of XML and SQL, and also basic PHP. DON'T PANIC if you don't know what these technologies are. We will cover in detail the exact XML you are expected to know, and the PHP api to RLIB (which is all you need to know). We won't cover SQL queries, except for the examples given.

RLIB is released under the GNU General Public License (GPL). This means that you are free to modify, copy, and redistribute it as long as you adhere to the terms of the GPL. If you or your organization do not want to make your code free, an alternative commercial license is available. You can review the license at http://www.gnu.org/copyleft/gpl.html.

RLIB is open source. Because of that, you might take the time to review the code and possibly make changes to improve it. SICOM Systems would be happy to review your changes, and possibly include them in future releases of RLIB. To find out more please visit the "Community" section on the RLIB web site.

What is a Report Writer

There are many report writers that exist, however the basic principles are the same. Loop through data sets printing out records in a specified format. Also allow for intelligent page breaking, report headers and footers, page headers and footers, allow for data grouping and breaking for subtotal. Be able to define variables that operate on columns (sum, average) that may reset on breaks. Also allow report columns to be mathamatical equations supporting a variety of functions.

RLIB supports all of this and more.

This Manual

The RLIB manual will cover all aspects of RLIB programming. Chapter 2 will cover RLIB terminology. Chapter 3 will cover rlib XML file format. Chapter 4 will cover RLIB data types and functions. Chapter 5 will discuss the PHP api to RLIB.

Chapter 2. Basic Training

A little Terminology

In addition to being a report writer, RLIB is also an interpreted language (like Basic and FOXPRO). RLIB handles all memory management for you. If you are familiar with FOXPRO, then you will feel at home with RLIB. However, if you don't know foxpro see the function reference (symbol table) to see all of the functions available to you. In a nutshell its pretty powerful. You can do stuff like this:

```
'hi' + ' ' + 'there' + database.stringfield
1+2*87+(7*2)/98.2/database.numberfield
dtos(database.somedatefield)-7
```

RLIB has 3 data types that it uses internally. STRING, DATE, and NUMBER. There are functions to turn datatypes into other datatypes, plus all sorts of other goodies. Almost all parameters in the RLIB xml file are expressions. This means that the parameters are computed on the fly. The result must be a STRING, DATE, or NUMBER (usually STRING or NUMBER). In our examples you will see stuff like expN, expS, expD. expN means number expressions, expS means a string expression, and expD means a date expression.

STRINGS are in single quotes

Please note that STRINGS in rlib are between single quotes. This is because XML uses double quotes.

lets take an example.

The round function is defined as round(expN). This means that it must be given a number to round. In rlib you an do the following:

```
round(7.2)
round(7.2+33)
round(val('7.2')*9/2)
```

As long as it gets passed a expN.. got it? Good. By the way, val(expS) takes a string and turns it into a number.

The DATE variable type

The RLIB DATE variables can hold either a date, a time or both. Mathematical operations on DATE variables are supported. When doing date/time arithmetic, the operations performed depend on whether the date component, time component or both are validly set to a value. For example, assuming date() returns a DATE variable and both time and date are set to 1/1/2004 at 10:31:45:

```
date() + 15
```

is 1/1/2004 at 10:32:00

```
chgdateof(date(), dateof(date()) + 15)
```

is 1/16/2004 at 10:31:45

```
timeof(date()) + 15
```

is 10:32:00

```
dateof(date()) + 15
```

is 1/16/2004

Fixed Point

Numerical accuracy is important in RLIB, therefore all mathematical calculations are done with fixed point using 8 decimal places of accuracy. As you might know, floating point operations in computing are inherently inaccurate. It is especially true if you are looping through data doing computations. Therefore, rlib uses FIXED POINT calculations internally in order to preserve accuracy. Note, all of this is internal to rlib, and you never have to worry about it when writing reports.

Data Sources

RLIB has many data sources; databases, environment variables, rlib internal variables, and rlib user variables.

The first query you give rlib is the main query. In the XML you can reference your fields as the field names without specifying the data source name ie (fld1+fld2). For the main data source, just put the field names in from the result set. You can still reference by the "RESULT SET NAME". Refer to the api "rlib_add_query_as".

Note

All Result set fields come into RLIB as STRINGS! you will have to frequently do VAL or FXPVAL. to turn them into numbers

All other queries are secondary queries. You **must** specify the data source name when you reference them. ie (result.fld1+result.fld2)

Environment variables are STRINGS coming from the environment that you can pick up to put on your report. If in php you had

```
$start_date = "2003-01-01";
```

In RLIB you reference thing as m.name. So you would have

```
m.start_date
```

rlib has a few internal variables like pageno, lineno, detailcnt (detail line count). Reference them with r.name. It would be r.pageno, r.lineno, and r.detailcnt in your code. RLIB has one more variable r.value, which is the result of a "FIELD"'s expression.. This is good so say in the color field, you DON'T HAVE TO RECALCULATE THE VALUE, you can use the "POINTER" to the value in order to color the field. like if it is < 0 make it red, else make it black....

Finally, rlib variables (discussed below) are referenced with a r.name, where name is the name you gave them.

So you could have an expression like this in rlib

```
str((val(m.some_environment_value)+r.some_sum_value*val(field1)),7,2) + ’ hi’
```

Report Layout

Reports are laid out into logical sections.

Report Header - Appears at the top of the first page of a report. Report headers commonly have information such as organization name.

Page Header - Appears at the top of every page except the first page, where it appears below the Report Header. Page headers commonly list the report name.

Page Footer - Appears at the bottom of each page. Page number is commonly found on Page Footers.

Report Footer - Appears on the last page below the last element of data. Report Data Totals are found in the Report Footer

Data Lines - Appear for each row of data you have in your main result set.

Break Header(s) - A header record for your data breaks appear before the Data Lines that are in the break sub section.

Break Footers(s) - Appear below the data lines for the break sub section. Might include sub totals for the data subsection.

Breaks Explained

It is often necessary to group data on a report in logical sections. For example imagine a report with a data scope of a list of stores. The stores are part of a hierarchy. In this example the hierarchy could be 4 levels deep. ie: "District, Market, Region, Another Level". It would be wasteful to have 4 columns on a report showing the same information over and over again. If the names were 20 characters long we would have just wasted at least 80 columns on our report. Aside from wasting space, it is often necessary to produce sub totals on groups of data in a report. Breaks are good for this also. The following example is a report that is breaking on a hierarchy schema. There are count and amount subtotals for each break. It should also be noted here that RLIB knows how much room is left on a page. It will not start a break at the bottom of a page if at least 1 data line will fit. Instead it will end the page and start the break on the next page.

Report Variables

Report Variables are are expressions rlib evaulates for every detail line, and resets automatically if asked to on breaks. Report variables are useful things like COUNT, SUM, AVG, HIGHEST, LOWEST. You may also use report variables to simplify expressions that will be used (normally more than once) in other calculations. To do this use the type EXPRESSION. Report variable can be "RESET" on the event of a BREAK (this does not apply to EXPRESSION). Here are some examples

```
<Variable name="customer_count" value="val(ei_count)+val(to_count)+val(dt_count)"
  type="expression"/>
<Variable name="daily_percent" value="v.daily_diff / v.this_year_netsales * 100"
  type="expression"/>
<Variable name="this_year_mtd" value="v.this_year_netsales"
  type="sum" resetonbreak="break5"/>
<Variable name="last_year_mtd" value="v.last_year_netsales"
  type="sum" resetonbreak="break5"/>
<Variable name="monthly_diff" value="v.this_year_mtd - v.last_year_mtd"
```

```
  type="expression"/>
<Variable name="monthly_percent" value="v.monthly_diff / v.this_year_mtd * 100"
  type="expression"/>
```

Defining a Report

RLIB reports are defined in the RLIB xml file. The XML file encompases all topics discussed above. Here is a sample XML file.

SAMPLE XML FILE

```
<?xml version="1.0"?>
<!DOCTYPE report >

<Report fontSize="9" orientation="landscape">
 <ReportHeader>
  <Output>
   <Image value="'logo.jpg'" type="'jpeg'" width="50" height="50"/>
   <Line/>
   <Line fontSize="12">
    <literal width="8"/>
    <field value="header.name" align="left" col="1"/>
   </Line>
   <Line fontSize="12">
    <literal width="8"/>
    <field value="header.name2" align="left" col="1"/>
   </Line>
   <Line/>
   <Line fontsize="4"/>
   <HorizontalLine size="4" bgcolor="'white'"/>
   <HorizontalLine size="2" bgcolor="'black'"/>
   <HorizontalLine size="4" bgcolor="'white'"/>
  </Output>
 </ReportHeader>

 <PageHeader>
  <Output>
   <Line fontSize="11">
    <field value="header.report_name" width="40" align="left" col="1"/>
   </Line>
   <HorizontalLine size="4" bgcolor="'white'"/>
  </Output>
 </PageHeader>

 <Detail>
  <FieldHeaders>
   <Output>
    <HorizontalLine size="1" bgcolor="'black'"/>
    <Line bgcolor="'0xe5e5e5'">
     <literal width="15" col="1">Number</literal>
     <literal width="1"/>
     <literal width="20" col="2">Name</literal>
     <literal width="1"/>
     <literal width="10" col="3">Type</literal>
     <literal width="1"/>
     <literal width="10" col="4">Category</literal>
    </Line>
    <HorizontalLine size="1" bgcolor="'black'"/>
    <HorizontalLine size="4" bgcolor="'white'"/>
   </Output>
  </FieldHeaders>
  <FieldDetails>
   <Output>
    <Line bgcolor="iif(r.detailcnt%2,'0xe5e5e5','white')">
     <field value="plunum" width="15" align="left" col="1"/>
     <literal width="1"/>
     <field value="name" width="20" align="left" col="2"/>
     <literal width="1"/>
     <field value="type" width="10" align="left" col="3"/>
     <literal width="1"/>
     <field value="category" width="10" align="left" col="4"/>
    </Line>
   </Output>
  </FieldDetails>
 </Detail>

 <PageFooter>
  <Output>
   <Line>
    <literal>Page: </literal>
    <field value="r.pageno" width="3" align="right"/>
   </Line>
  </Output>
 </PageFooter>

 <ReportFooter>
 </ReportFooter>
</Report>
```

Report

All Reports are made up of the Report tag.

fontSize - Font size .. 6-100

orientation - "portrait" or "landscape"

topMargin - how much space to leave at the top

leftMargin - how much space to leave at the left

bottomMargin - how much space to leave at the bottom

paperType - defines the type of paper you are targeting with your report. This field is a STRING Expression. values are: 'LETTER', 'LEGAL', 'A4', 'B5', 'C5', 'DL', 'EXECUTIVE', 'COMM10', 'MONARCH', and 'FILM35MM'

Output

All main sections of a report contain output. Output is made up of Images, Lines, and Horizontal Lines.

HorizontalLine

bgcolor - background color

size - how tall the line is

indent - offset the line from the left margin (NUMBER VALUE.. number of characters) length - the length of the line (NUMBER VALUE.. number of characters)

fontSize - part of the math on the length of and indent of a line. This is here for finer control because indent and length are given in number of characters.

Image

value - expS with name of file [REQUIRED]

type - expS with type of file (jpeg, gif) [REQUIRED]

width - expN width of image [REQUIRED]

height - expN with height of image [REQUIRED]

Line

Lines contain literals and fields.

fontSize - expN default font size for literals and fields on the line

color - expS default foreground color for literals and fields on the line

bgcolor - expS default background color for fields on the line

Literals

Contains plain text data. The actual displayed text goes between the tag

width - expN width of string

align - expS left, right, center...

color - expS default foreground color

col - expN column for csv output

bgcolor - expS background color

link - expS .. a url to link to

Field

Contains Data

value - EXPRESSION of any kind

col - expN column for csv output

width - expN width of string

align - expS left, right, center

format - expS see format section

color - expS foreground color

bgcolor - expS background color

link - expS .. a url to link to

ReportHeader

Contains Output

ReportFooter

Contains Output

PageHeader

Contains Output

PageFooter

Contains Output

Detail

Contains Field Headers and Field Details. Headers are essentially the column headers. Field details are the lines repeated over and over again.

Variables

Variables are used to manuiplate data on the floor and save it. Variables can sum, count, average, or can be simple expressions. Variables have a name, a type and optionally when they should reset their value.

name - expS the name of the variable, referenced in expressions as v.name [REQUIRED]

value - Any expression. If it is a sum, average, count it must be a expN

resetonbreak - name of break that the variable should reset

Breaks

Breaks are hierarchial groupings of data on a report.

name - expS the name of the variable, referenced in expressions as v.name [REQUIRED]

value - Any expression If it is a sum, average, count it must be a expN

newpage = [yes/no] No is default should the break start on a new page

headernewpage = [yes/no] Yes is default. The break header will always appear on the top of every page

breaks have BreakHeader and BreakFooter and both contain output. ONE NICE FEATURE IN RLIB IS THAT WHEN A BREAK EVENT HAPPENS THE OUTPUT will be done while on current row (FOR FOOTER AND HEADER). This makes subtotals much easier!

Breaks also have BreakFields. An expression on how it breaks. There can be more then one and it has a value. The value is an expression.

Chapter 3. RLIB Function Table

In the definitions below; expN means a number expressions, expS means a string expression, and expD means a date expression.

+

expN1 + expN2 **Returns** NUMBER containing the result of expN1 plus expN2

expD + expN or expN + expD **Returns** DATE contains the result of expD + expN seconds or days. If both datetimes have valid time, the result is expressed in SECONDS. Otherwise the result returned is in DAYS.

expS1 + expS2 **Returns** STRING containing the result of the concatination of expS1 and expS2

-

expN1 - expN2 **Returns** NUMBER contain the result of expN1 minus expN2

expD - expN **Returns** if DATE contains a valid time portion then the result is expD - expN SECS. Otherwise it returns expD - expN DAYS.

expD - expD **Returns** NUMBER containing the number of SECONDS or DAYS differnce in the dates. If the time portion is valid, the result is the number of SECONDS difference. If both times and both dates are valid the seconds count will include the difference in the days. Otherwise if the date portions are valid it returns the number of DAYS difference.

*

expN1 * expN2 **Returns** NUMBER containing the result of expN1 times expN2

/

expN1 / expN2 **Returns** NUMBER containing the result of expN1 divided by expN2

%

expN1 % expN2 **Returns** NUMBER containing the result of expN1 mod expN2

^

expN1 ^ expN2 **Returns** NUMBER containing the result of expN1 to the expN2

<=

expN1 <= expN1 **Returns** NUMBER 1 if true.. 0 if false

expS1 <= expS2 **Returns** NUMBER 1 if true if expS1 is alphebatically less then expS2.. 0 if false

expD1 <= expD2 **Returns** NUMBER 1 if true if expD1 is less then expD2.. 0 if false. Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

<

expN1 < expN2 **Returns** NUMBER 1 if true.. 0 if false

expS1 < expS2 **Returns** NUMBER 1 if true.. 0 if false

expD1 < expD2 **Returns** NUMBER 1 if true.. 0 if false Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

>=

expN1 >= expN2 **Returns** NUMBER 1 if true.. 0 if false

expS1 >= expS2 **Returns** NUMBER 1 if true.. 0 if false

expD1 >= expD2 **Returns** NUMBER 1 if true.. 0 if false Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

>

NUMBER > NUMBER **Returns** NUMBER 1 if true.. 0 if false

STRING > STRING **Returns** NUMBER 1 if true.. 0 if false

expD1 > expD2 **Returns** NUMBER 1 if true.. 0 if false Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

==

NUMBER == NUMBER **Returns** NUMBER 1 if true.. 0 if false

STRING == STRING **Returns** NUMBER 1 if true.. 0 if false

expD1 == expD2 **Returns** NUMBER 1 if true.. 0 if false Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

!=

NUMBER != NUMBER **Returns** NUMBER 1 if true.. 0 if false

STRING != STRING **Returns** NUMBER 1 if true.. 0 if false

expD1 != expD2 **Returns** NUMBER 1 if true.. 0 if false Both date and time are compared if both have valid components. The dateof and timeof functions can be used to select one or the other.

&&

LOGICAL "AND". iif(m.test1 == 22 && m.test2 == 17, ’passed’, ’failed’)

NUMBER && NUMBER **Returns** NUMBER 1 if true.. 0 if false

STRING && STRING **Returns** NUMBER 1 if true.. 0 if false .. True if both strings are not null

||

NUMBER || NUMBER **Returns** NUMBER 1 if true.. 0 if false

STRING || STRING **Returns** NUMBER 1 if true.. 0 if false . True if at least 1 of the strings is not null

abs(expN)

Absolute Value.

Returns NUMBER

ceil(expN)

Round up to the nearest integer.

Returns NUMBER

floor(expN)

Round down to the nearest integer.

Returns NUMBER

round(expN)

Round to nearest integer

Returns NUMBER

sin(expN)

The sin() function Returns the sine of expN, where expN is given in radians.

Returns NUMBER

cos(expN)

The cos() function Returns the cosine of expN, where expN is given in radians.

Returns NUMBER

ln(expN)

The ln() function Returns the natural logarithm of of expN.

Returns NUMBER

exp(expN)

The exp() function Returns the value of e (the base of natural logarithms) raised to the power of expN

Returns NUMBER

atan(expN)

The atan() function calculates the arc tangent of expN; that is the value whose tangent is expN.

Returns NUMBER

sqrt(expN)

The sqrt() function Returns the non-negative square root of expN. Don't pass it a negative number or it will fail I think

Returns NUMBER

val(expS)

Converts a string into a number.. respects the decimal point in a string

Returns NUMBER

fxpval(expS, expN)

Converts a string into a number.. assumes no decimal places in the string.. expN says where the decimal place should go for instance if in your database you store all values in cents.. and not dollars

Returns NUMBER

str(expN1, expN2, expN3)

Convert a NUMBER into a STRING of length expN2 with decimal percision of expN3

Returns STRING

stod(expS)

Converts a string into a date.. must be int he format of YYYY-MM-DD (like sql)

Returns DATE

iif(exp1, exp2, exp3)

In-line If. Basically exp1 is evaulated. If it is true (NUMBER != 0, STRING != NULL, DATE is always true).. exp2 is evaulated and returned otherwise exp3 is evaluated and returned

Returns exp2 if TRUE... exp3 if FALSE

dtos(expD)

Converts a date into a string. The string will be YYYY-MM-DD

Returns STRING

year(expD)

Returns YYYY as NUMBER

Returns NUMBER

month(expD)

Returns MM as NUMBER

Returns NUMBER

day(expD)

Returns DD as NUMBER

Returns NUMBER

upper(expS)

Returns expS as a upper case string

Returns STRING

lower(expS)

Returns expS as a lower case string

Returns STRING

proper(expS)

Returns expS as a proper string.. ie 1str char caps.. all the rest lower

Returns STRING

stodt(expS)

Converts a string into a date time.. must be int he format of YYYYMMDDHHMMSS (like sql)

Returns DATE

isnull(expS)

Returns expN 1 if expS is null or expN 0 if expS is not null

Returns NUMBER

dim(expD)

Returns NUMBER containting the day in the month (1-31)

Returns NUMBER

wiy(expD)

Returns NUMBER containting the week number of the year. Range 00 to 53, starting with the first Sunday as the first day of week 01

Returns NUMBER

wiyo(expD, expN)

Returns NUMBER containing the week number of the year. Range 00 to 53, starting with the first expN as the first day (1=Monday, 2=Tuesday...)

Returns NUMBER

date()

Returns the current date time in RLIB DATE Format. This value is set to a constant when report generation starts and remains unchanged for the duration of the report. **This is a change from previous versions of RLIB which would reset this value to the current time on each invocation.** Locking this value prevents the current date/time from changing from page to page when the current date/time is repeated on multiple pages.

Returns DATE

left(expS, expN)

Returns the leftmost expN characters of the string expS.

Returns STRING

right(expS, expN)

Returns the rightmost expN characters of the string expS.

Returns STRING

mid(expS, expN1, expN2)

expN1 is an index into the string expS starting at 0. expN2 is the maximum length of the result string.

Returns STRING

tstod(expS)

expS is a time in string format such as HH:MM, HH:MM:SS, HH:MMp, HH:MM:SSp, HHMM, HHMMSS, HHMMp, HHMMSSp. This function will return a DATE value with the time set to the time in the string (the date part of the DATE variable is set to 1/1/1980).

Returns DATE

dtosf(expD, expS)

DEPRECATED! This function may be removed in future versions of RLIB. Please use the 'format' function instead.

Convert date expD to string using the specified format expS. expS uses the formatting controls listed below and can be used to display either the date or time of the DATE variable.

Returns STRING

format(exp_, expS)

Formats the passed exp_ (may be expD, expS or expN) using the expS parameter as a format string. Format strings **must** be in the new '!' format.

This function will format the passed variable using the passed format string. Character strings may be formatted using the format symbol '!!'. Errors will be generated if the type for the format string does not match the variable type. format headers are: '!!' for strings, '!@' for datetimes, '!#' for numbers and '!$' for currency.

For example: "format('xyx', '!!%6s')" prints ' xyz'.

Returns STRING

true

A predefined NUMBER variable with the value 1.

Returns NUMBER

yes

A predefined NUMBER variable with the value 1.

Returns NUMBER

false

A predefined NUMBER variable with the value 0.

Returns NUMBER

no

A predefined NUMBER variable with the value 0.

Returns NUMBER

dateof(expD)

A function that converts a datetime to a date only variable. Use this function with comparisons and DATE arithmetic to select only the DATE portion for use in the expression.

Returns DATE

timeof(expD)

A function that converts a datetime to a time only variable. Use this function with comparisons and TIME arithmetic to select only the TIME portion for use in the expression.

Returns DATE

chgdateof(expD1, expD2)

Changes the date portion of expD1 to equal that of expD2.

Returns DATE

chgtimeof(expD1, expD2)

Changes the time portion of expD1 to equal that of expD2.

Returns DATE

gettimeinsecs(expD)

Returns the time portion of the datetime variable as the number of seconds past 00:00:00

Returns NUMBER

This function in conjunction with settimeinsecs can be used to perform mathematical calculations on time.

settimeinsecs(expD, expN)

Returns a datetime with the time portion of the datetime changed to a value obtained by adding expN seconds to 00:00:00. For example: "settimeinsecs(expD, (gettimeinsecs(expD) / 3600) * 3600)" returns a datetime that has been truncated to an even hour value.

Returns DATE

see settimeinsecs function for details and example.

Chapter 4. Formatting

Foreground Color

color = "exprS"

exprS must be either a valid color name or hex color triplet ie 0xFFFFFF.

Use the forground color to change the color for literals and fields

Background Color

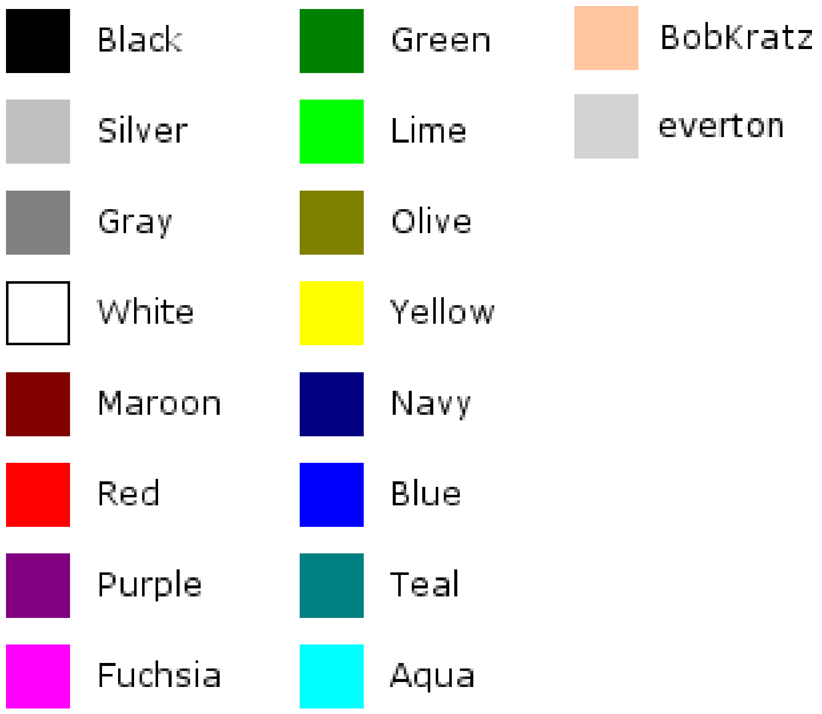
bgcolor = "exprS"

exprS must be either a valid color name or hex color triplet ie 0xFFFFFF.

Use the background color to change the color for literals and fields

COLORS IN GENERAL

Colors must be a STRING an specified as either a named color. See chart for color names.


Black
Silver
Gray
White
Maroon
Red
Purple
Fuchsia
Green
Lime
Olive
Yellow
Navy
Blue
Teal
Aqua
BobKratz
everton

OR as a hex color tripplet like 0xFFFFFF

so in the XML it would be either color="’0xFFFFFF’" OR color="’red’"

Format Strings

Yup RLIB has them. They are very similar to C. Numbers, Dates, or Strings. Here is the neat part. If you mess up format strings, rlib will put an error message in the field. They are...

!ERR_F means that rlib was not given a format string and can’t automatically make something of your data

!ERR_F_D means you asked rlib to format as a number but it was not given a number data type

!ERR_F_S means you asked rlib to format as a string but it was not given a string data type

!ERR_F_F means you asked rlib could not interperate your format string expression / or what it interpreted it to was not a string

It works a lot like c. You can do stuff like "'You have %d apples'"

If you do something like %$5.2d this means put commas in so you will get 12,345.67

New Style Format Strings with "!" prefix

The new style format strings can be intermixed with old style formatting and may be used anywhere a format string is needed. They provide locale aware formatting for date/time, money and numbers. The first 2 characters in the new format strings must begin with either "!@", "!$" or "!#", respectively. Following this is an appropriate 'C' style format string for one of the functions strftime, strfmon, or sprintf (for numbers). All numerics in the money and number format strings must be represented using the 'e', 'f' or 'g' format codes. For example "!#$.2g", "!$%n", "!@%m/%d/%Y" are valid format strings using the new style for a number, a money amount and a date. There are also additional error codes added as follows:

!ERR_DT_D means you asked rlib to format a date but the date field of the datetime is not set.

!ERR_DT_T means you asked rlib to format a time but the time field of the datetime is not set.

!ERR_DT_NO means there were no valid format codes in a format string for a date/time.

String Format Strings

Exactly like c. %[optional number]s where optional number is how big to make the string

Number Format Strings

Close to c. %[optional number 1][.][optional number 2]d ... where optional number 1 is how big should the left side be and optional number 2 is how many decimal places...

Number Format Strings (!# format)

Identical to C. All numbers must be represented by the e, f or g format types.

Please refer to your systems printf formatting codes.

Date Format Strings (all including !@ format)

Date codes and time codes should be consecutive, i.e. don't have a timecode a datecode and then another timecode. This will not work. Rlib internally splits the datetime string into a date format string and a time format string. It uses the first transition from date-to-time or time-to-date as the split point for the date/time.

Dates from 1/1/1 through 1/1/8000+ can be represented.

%a The abbreviated weekday name according to the current locale.

%A The full weekday name according to the current locale.

%A The full weekday name according to the current locale.

%b The abbreviated month name according to the current locale.

%B The full month name according to the current locale.

%c The preferred date and time representation for the current locale.

%C The century number (year/100) as a 2-digit integer. (SU)

%d The day of the month as a decimal number (range 01 to 31).

%D Equivalent to %m/%d/%y. (Yecch - for Americans only. Americans should note that in other coun- tries %d/%m/%y is rather common. This means that in international context this format is ambigu- ous and should not be used.) (SU)

%e Like %d, the day of the month as a decimal number, but a leading zero is replaced by a space. (SU)

%E Modifier: use alternative format, see below. (SU)

%F Equivalent to %Y-%m-%d (the ISO 8601 date format). (C99)

%G The ISO 8601 year with century as a decimal number. The 4-digit year corresponding to the ISO week number (see %V). This has the same format and value as %y, except that if the ISO week num- ber belongs to the previous or next year, that year is used instead. (TZ)

%g Like %G, but without century, i.e., with a 2-digit year (00-99). (TZ)

%h Equivalent to %b. (SU)

%H The hour as a decimal number using a 24-hour clock (range 00 to 23).

%I The hour as a decimal number using a 12-hour clock (range 01 to 12).

%j The day of the year as a decimal number (range 001 to 366).

%k The hour (24-hour clock) as a decimal number (range 0 to 23); single digits are preceded by a blank. (See also %H.) (TZ)

%l The hour (12-hour clock) as a decimal number (range 1 to 12); single digits are preceded by a blank. (See also %I.) (TZ)

%m The month as a decimal number (range 01 to 12).

%M The minute as a decimal number (range 00 to 59).

%n A newline character. (SU)

%O Modifier: use alternative format, see below. (SU)

%p Either ‘AM’ or ‘PM’ according to the given time value, or the corresponding strings for the cur- rent locale. Noon is treated as ‘pm’ and midnight as ‘am’.

%P Like %p but in lowercase: ‘am’ or ‘pm’ or a corresponding string for the current locale. (GNU)

%r The time in a.m. or p.m. notation. In the POSIX locale this is equivalent to ‘%I:%M:%S %p’. (SU)

%R The time in 24-hour notation (%H:%M). (SU) For a version including the seconds, see %T below.

%s The number of seconds since the Epoch, i.e., since 1970-01-01 00:00:00 UTC. (TZ)

%S The second as a decimal number (range 00 to 61).

%t A tab character. (SU)

%T The time in 24-hour notation (%H:%M:%S). (SU)

%u The day of the week as a decimal, range 1 to 7, Monday being 1. See also %w. (SU)

%U The week number of the current year as a decimal number, range 00 to 53, starting with the first Sunday as the first day of week 01. See also %V and %W.

%V The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week. See also %U and %W. (SU)

%w The day of the week as a decimal, range 0 to 6, Sunday being 0. See also %u.

%W The week number of the current year as a decimal number, range 00 to 53, starting with the first Monday as the first day of week 01.

%x The preferred date representation for the current locale without the time.

%X The preferred time representation for the current locale without the date.

%y The year as a decimal number without a century (range 00 to 99).

%Y The year as a decimal number including the century.

%z The time-zone as hour offset from GMT. Required to emit RFC822-conformant dates (using "%a, %d %b %Y %H:%M:%S %z"). (GNU)

%Z The time zone or name or abbreviation.

%+ The date and time in date(1) format. (TZ)

%% A literal '%' character.

Money Format String (!$ format)

The money formatter is locale aware and will use appropriate symbols and formats for the designated locale. It is a string in the form: %[=f ^ (+ ! -][fieldwidth][#leftprecision][.rightprecision][n or i] [] indicates optional sections.

=f f is a character to use as the numeric fill character. Default is ' '.

^ ignore grouping if specified in the locale. This is usually thousands groupings.

(Put negative amounts in ().

+ Show + sign on positive numbers.

! Omit the currency symbol.

- Left justify all fields.

n Display in national format. Like: $1.25

i Display in international format. Like USD 1.25

%% represents a % sign within the format specification string.

Default Format Strings

For STRING it is "%s"

for NUMBER it is "%d"

for DATE it is "%m/%d/%Y"

Chapter 5. C API

rlib_init_with_environment

rlib * rlib_init_with_environment(struct environment_filter *environment)

Create an instance of RLIB. You will normally pass NULL to init. However.. at some point we need to better document what the environment_filter does... this is normally used for 3rd part bindings like PHP, PERL, or PYTHON.

Returns a pointer to a **rlib**

rlib_init

rlib * rlib_init()

calls rlib_init_environment with a NULL pointer

Returns a pointer to a **rlib**

rlib_add_datasource_mysql

rlib_add_datasource_mysql(**rlib * rlib_ptr**, char *datasource_name, char *hostname, char *username, char *password, char *database)

Add a mysql datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with.

This function is only available if rlib is compiled with mysql support.

rlib_add_datasource_postgre

rlib_add_datasource_postgre(**rlib * rlib_ptr**, char *datasource_name, char *connection_string)

Add a postgre datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with. The connection_string is the standard postgre connection string.. which might contain user and password information, among other things.

This function is only available if rlib is compiled with postgre support.

rlib_add_datasource_odbc

rlib_add_datasource_odbc(**rlib * rlib_ptr**, char *datasource_name, char *user_name, char *password)

Add a odbc datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with. The user name and password are that of your database you are connecting to

This function is only available if rlib is compiled with odbc support.

rlib_datasource_set_decoding

rlib_datasource_set_decoding(**rlib * rlib_ptr**, char *datasource_name, char *decoding)

Set the decoding method for your datasource if its not already ISO8859-1

rlib_add_query_as

rlib_add_query_as(**rlib * rlib_ptr**, char *datasource_name, char *query, char *rlib_query_name)

The query is added to an execution queue, but it is not executed at this time. The name is important because you can reference result sets directly in your rlib xml files. The first query added is assumed to be the main loop query. The datasource name must match a datasource the your provided rlib, such as one of the mysql or postgre datasources.

rlib_add_report

rlib_add_report(**rlib *rlib_ptr**, char *rlib_xml_file, char *rlib_query_name)

A report is added to the report execution queue but not compiled at this time.

rlib_query_name [OPTIONAL] - The name of the rlib_query to use in the main loop of the report. Pass NULL if you don't need to specify the query name

rlib_set_output_format

rlib_set_output_format(**rlib *rlib_ptr**, int type)

Type can be one of the following: RLIB_FORMAT_PDF, RLIB_FORMAT_HTML, RLIB_FORMAT_TXT, RLIB_FORMAT_CSV

rlib_set_output_format_from_text

rlib_set_output_format_from_text(**rlib *rlib_ptr**, char *name)

Type can be one of the following: "pdf", "html", "csv", "txt"

rlib_execute

rlib_execute(**rlib *rlib_ptr**)

Connects to the database, runs queries, compiles xmls and buffers up a report.

rlib_get_content_type

char *rlib_get_content_type(**rlib *rlib_ptr**)

This will return a string content type Use it with the **php header function**. Even if you ask for a PDF you might not get a PDF because errors might occur. If this is the case, rlib defaults to html and sends out error messages.

rlib_spool

rlib_spool(**rlib * rlib_ptr**)

Rlib will send the output out stdout.

rlib_get_output

char * rlib_get_output(**rlib * rlib_ptr**)

Returns the output buffer (COULD BE NON NULL TERMINATED STRING)

rlib_get_output_length

long rlib_get_output_length(**rlib * rlib_ptr**)

Returns the length of the output buffer

rlib_add_parameter

int rlib_add_parameter(**rlib *rlib_ptr**, const char *name, const char *value)

Adds the name/value pair to the memory parameters. Values added in this manner supercede values passed in the environment. The names are searched in a case sensitive manner. Both name and value are stored by value, so the passed arguments do not need to persist after the call.

rlib_free

rlib_free(**rlib *rlib_ptr**)

Free rlib's memory that it allocated

rlib_add_resultset_follower

rlib_add_resultset_follower(**rlib *rlib_ptr**, char *leader, char *follower)

Adds the ability to have more then one main loop query. leader and follower are the names of the queries you set in rlib_add_query_as.

rlib_version

char *rlib_version(void);

Returns a string containing the version of RLIB being used.

rlib_set_output_encoding(rlib *rlib_ptr, const char *encoding)

Sets the output character encoding, overriding any encoding that is set in the current Locale. By default RLIB will use the character encoding indicated in the current Locale settings. All reports will use this encoding unless overriden by a call to rlib_set_report_output_encoding. If the encoding is NULL, or a null string, the output will be left in UTF-8 encoding.

rlib_set_report_output_encoding(rlib *rlib_ptr, int reportnumber, const char *encoding)

Sets the output character encoding for the indicated report. This setting will override the default setting. If this is not set or encoding is NULL or a null string, the **default rlib encoding is used**.

rlib_set_pdf_font(rlib *rlib_ptr, const char *encoding, const char *fontname)

Sets the output character encoding for the indicated report. If this is not set or encoding is NULL or a null string, the **default rlib encoding is used**.

rlib_set_locale(rlib *rlib_ptr, const char *locale)

Sets the locale to the passed locale. The locale must be one of the values returned by the shell command: **locale -a**.

Returns true if the locale was successfully set.

SAMPLE

Here is a example.

```
#include <rlib.h>

char *query ="SELECT * FROM plu";
rlib *r;

r = rlib_init();
rlib_add_datasource_mysql(r, "mysql", "localhost", "user", "password", "database");
rlib_add_query_as(r, "mysql", query, "woot");
rlib_add_report(r, "report.xml");
rlib_set_output_format(r, $format);
rlib_execute(r);
rlib_spool(r);
rlib_free(r);
```

Chapter 5. PHP API

rlib_init

rlib_init()

Start RLIB

Returns a pointer to a **rlib**

rlib_add_datasource_mysql

rlib_add_datasource_mysql(**rlib**, datasource_name, hostname, username, password, database)

Add a mysql datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with.

This function is only available if rlib is compiled with mysql support.

rlib_add_datasource_postgre

rlib_add_datasource_postgre(**rlib**, datasource_name, connection_string)

Add a postgre datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with. The connection_string is the standard postgre connection string.. which might contain user and password information, among other things.

This function is only available if rlib is compiled with postgre support.

rlib_add_datasource_odbc

rlib_add_datasource_odbc(**rlib**, datasource_name, user_name, password)

Add a odbc datasource. The datasource_name is used in rlib_add_query_as to tell rlib which datasource to run the query with. The user name and password are that of your database you are connecting to

This function is only available if rlib is compiled with odbc support.

rlib_datasource_set_decoding

rlib_datasource_set_decoding(**rlib**, datasource_name, decoding)

Set the decoding method for your datasource if its not already ISO8859-1

rlib_add_query_as

rlib_add_query_as(**rlib**, datasource_name, query, rlib_query_name)

The query is added to an execution queue, but it is not executed at this time. The name is important because you can reference result sets directly in your rlib xml files. The first query added is assumed to be the main loop query. The datasource name must match a datasource the your provided rlib, such as one of the mysql or postgre datasources.

rlib_add_report

rlib_add_report(**rlib**, rlib_xml_file, [rlib_query_name])

A report is added to the report execution queue but not compiled at this time.

rlib_query_name [OPTIONAL] - The name of the rlib_query to use in the main loop of the report

rlib_set_output_format_from_text

rlib_set_output_format_from_text(**rlib**, type)

Type can be one of the following: html, pdf, txt, or csv.

rlib_execute

rlib_execute(**rlib**)

Connects to the database, runs queries, compiles xmls and buffers up a report.

rlib_get_content_type

rlib_get_content_type(**rlib**)

This will return a string content type Use it with the **php header function**. Even if you ask for a PDF you might not get a PDF because errors might occur. If this is the case, rlib defaults to html and sends out error messages.

rlib_spool

rlib_spool(**rlib**)

Rlib will send the output out stdout.

rlib_add_parameter

int rlib_add_parameter(**rlib**, namestring, valuestring)

Adds the name/value pair to the memory parameters. Values added in this manner supercede values passed in the environment. The names are searched in a case sensitive manner.

rlib_free

rlib_free(**rlib**)

Free rlib's memory that it allocated

rlib_add_resultset_follower

rlib_add_resultset_follower(**rlib**, leader, follower)

Adds the ability to have more then one main loop query. leader and follower are the names of the queries you set in rlib_add_query_as.

rlib_version

rlib_version()

Returns a string containing the RLIB version number or the libarary.

rlib_set_output_encoding(rlib, encodingstring)

Sets the output character encoding, overriding any encoding that is set in the current Locale. By default RLIB will use the character encoding indicated in the current Locale settings. All reports will use this encoding unless overriden by a call to rlib_set_report_output_encoding. If the encoding is NULL, or a null string, the output will be left in UTF-8 encoding.

rlib_set_report_output_encoding(rlib, reportnumber, encodingstring)

Sets the output character encoding for the indicated report. This setting will override the default setting. If this is not set or encoding is NULL or a null string, the **default rlib encoding is used**.

rlib_set_pdf_font(rlib, encodingstring, fontnamestring)

Sets the output character encoding for the indicated report. If this is not set or encoding is NULL or a null string, the **default rlib encoding is used**.

rlib_set_locale(rlibr, locale string)

Sets the locale to the passed locale. The locale must be one of the values returned by the shell command: **locale -a**.

Returns true if the locale was successfully set.

SAMPLE

Here is a example.

```
dl ("librlib.so");
$query = "SELECT * FROM plu";
$rlib = rlib_init();
rlib_add_datasource_mysql($rlib, "mysql", "localhost", "user", "password", "database");
$format = "PDF";
rlib_add_query_as($rlib, "mysql", $query, "topline");
rlib_add_report($rlib, "report.xml");
rlib_set_output_format_from_text($rlib, $format);
rlib_execute($rlib);
header(rlib_get_content_type($rlib));
rlib_spool($rlib);
rlib_free($rlib);
```

Chapter 6. Examples

Example 1

It is assumed that you have a working PHP, APACHE, and MySQL setup. You will have to substitute the MySQL host, username, password, and database to what ever you created. The example is in three sections. First creating the database. Second we create the PHP source. Third we create the RLIB XML file.

MySQL Table Creation

```
DROP TABLE IF EXISTS example;

CREATE TABLE example (
 rn INT NOT NULL AUTO_INCREMENT,
 name VARCHAR(30) NOT NULL DEFAULT "",
 type INT NOT NULL,
 price FLOAT NOT NULL,
 PRIMARY KEY (rn),
 KEY (name)
);

INSERT INTO example (name, type, price)
 VALUES
 ("Hammer", 1, 10.00),
 ("Screw Driver", 1, 7.00),
 ("Bolts", 1, 2.00),
 ("Hot Dog", 2, 1.50),
 ("Soda", 2, 1.00),
 ("Chips", 2, 1.00),

 ("Jaguar", 3, 50000.00),
 ("Lexus", 3, 60000.00),
 ("Pinto", 3, 2000.00);
```

PHP Source

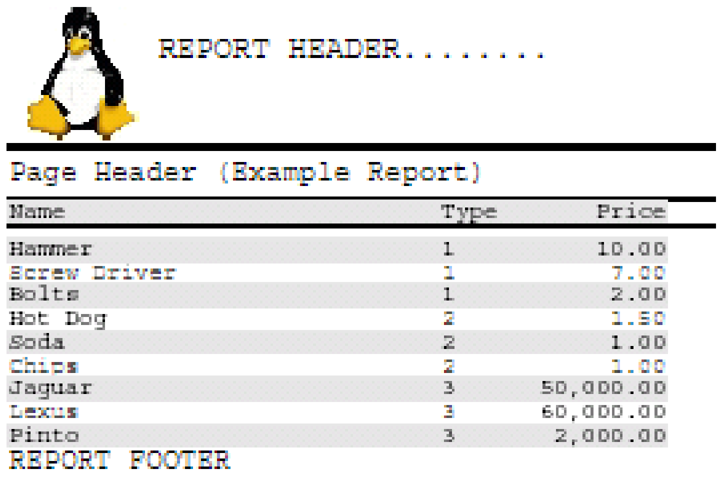
```
<? dl ("librlib.so");

 $format = "pdf";
 $sql_host = "localhost";
 $sql_users = "username";
 $sql_password = "password";
 $sql_database = "tablename";

 $rlib = rlib_init();
    rlib_add_datasource_mysql($rlib, "mysql", $sql_host, $sql_users, $sql_password, $sql_databas
    rlib_add_query_as($rlib, "mysql", "select * from example", "example");
    rlib_add_report($rlib, "report.xml");
    rlib_set_output_format_from_text($rlib, $format);
    rlib_execute($rlib);
    header( rlib_get_content_type($rlib));
    rlib_spool($rlib);
    rlib_free($rlib);
?>
```

RLIB XML SOURCE

```
<?xml version="1.0"?>
```

```
<!DOCTYPE report >
<Report fontSize="9" orientation="landscape">
 <ReportHeader>
  <Output>
   <Image value="'logo.jpg'" type="'jpeg'" width="50" height="50"/>
   <Line/>
   <Line fontSize="12">
    <literal width="8"/>
    <literal>REPORT HEADER........</literal>
   </Line>
   <Line/>
   <Line/>
   <Line fontsize="4"/>
   <HorizontalLine size="4" bgcolor="'white'"/>
   <HorizontalLine size="2" bgcolor="'black'"/>
   <HorizontalLine size="4" bgcolor="'white'"/>
  </Output>
 </ReportHeader>

 <PageHeader>
  <Output>
   <Line fontSize="11">
    <literal>Page Header (Example Report)</literal>
   </Line>
   <HorizontalLine size="4" bgcolor="'white'"/>
  </Output>
 </PageHeader>

 <Detail>
  <FieldHeaders>
   <Output>
    <HorizontalLine size="1" bgcolor="'black'"/>
    <Line bgcolor="'0xe5e5e5'">
     <literal width="30" col="1">Name</literal>
     <literal width="1"/>
     <literal width="5" col="2">Type</literal>
     <literal width="1"/>
     <literal width="10" col="3" align="right">Price</literal>
    </Line>
    <HorizontalLine size="1" bgcolor="'black'"/>
    <HorizontalLine size="4" bgcolor="'white'"/>
   </Output>
  </FieldHeaders>
  <FieldDetails>
   <Output>
    <Line bgcolor="iif(r.detailcnt%2,'0xe5e5e5','white')">
     <field value="name" width="30" align="left" col="1"/>
     <literal width="1"/>
     <field value="type" width="5" align="left" col="2"/>
     <literal width="1"/>
     <field value="val(price)" width="10" format="'%$.2d'" align="right" col="3"/>
    </Line>
   </Output>
  </FieldDetails>
 </Detail>

 <PageFooter>
  <Output>
   <Line>
    <literal>Page: </literal>
    <field value="r.pageno" width="3" align="right"/>
   </Line>
  </Output>
 </PageFooter>

 <ReportFooter>
```

```
  <Output>
   <Line fontSize="11">
    <literal>REPORT FOOTER</literal>
   </Line>
  </Output>
 </ReportFooter>
</Report>
```



```
REPORT HEADER.........

Page Header (Example Report)
Name                                Type         Price
Hammer                              1            10.00
Screw Driver                        1             7.00
Bolts                               1             2.00
Hot Dog                             2             1.50
Soda                                2             1.00
Chips                               2             1.00
Jaguar                              3        50,000.00
Lexus                               3        60,000.00
Pinto                               3         2,000.00
REPORT FOOTER
```

Appendix A. GNU Free Documentation License

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text

VERBATIM COPYING

COPYING IN QUANTITY

COMBINING DOCUMENTS

COLLECTIONS OF DOCUMENTS

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B. SWIG License

LICENSE

RLIB is distributed with language bindings that allow you to use rlib (the c library) in other languages. Some of the bindings were created using SWIG (www.swig.org). SWIG generated source is not GPL, however is still is free software.

Simplified Wrapper and Interface Generator (SWIG)

SWIG is distributed under the following terms: