
BEEBUG
Stand Alone Generator

C

Stand Alone Generator & Maths Functions

User Guide

Beebug

This user guide was written by John Wallace and David Allison.

FIRST EDITION 1987

Published by Beebug Limited, Dolphin Place, Holywell Hill,
St. Albans, Herts. ALl lEX, England. Telephone (0727) 40303

Contents

1. Introduction

The accompanying dual-format disc contains the Beebug C Stand Alone Generator, and a new version of the library rtlib containing a range of maths functions.

Normally the Beebug C Linker produces code which is only executable from within C, by the run-time interpreter. This means that in order to execute programs produced by the compiler, the Beebug C ROMs must be installed in the computer. The Stand Alone Generator provides the facility to produce code which may be run as a 6502 machine-code program without any special hardware or firmware requirements. This allows C programs to be executed from any language, and on any BBC or Master computer even if C is not installed (memory permitting). The Linker was designed with this facility in mind, and it is only necessary to have a special file called *rtsys* in the current library directory in order to use it.

The new library *rtlib* contains the following additional mathematical functions:

```
SIN COS TAN LOG LOGlO EXP POW RAD DEG FTOI IPOW
SQRT LDEXP MODF CElL FABS FLOOR FMOD FPEXP ITOF
```

A full description of each function is given in section 4 of this user guide. To use these new functions the new files *rtlib* and *h.math* must replace the earlier versions on your program disc.

2. Getting Started

The Stand Alone Generator disc contains the following files:

rtsys	-	Stand Alone Generator (**R**un **T**ime **SYS**tem)
rtlib	-	new function library (**R**un **T**ime **LIB**rary)
h.math	-	new mathematics header file
c.sinwave	-	mathematics example program

They should be copied to your C program disc, replacing the earlier versions (in the case of *rtlib* and *h.math*), Please note that *rtsys* and *rtlib* must be copied into the library directory of your C program disc (this is usually `$`).

Please note that the Stand Alone Generator disc is produced using a special dual-format, and cannot be backed-up in the usual way with `*BACKUP`. To make a copy of the disc use the `*COPY` command to copy all files in directories `$`, `H` and `C`.

3. Stand Alone Generator

The `STANDALONE` qualifier

To produce C stand-alone code, simply compile the program in the usual way, then link using the **`STANDALONE`** qualifier. For example:

```
COMPILE welcome
LINK/STANDALONE welcome
```

will produce stand-alone code for the program welcome. As usual the executable code will be stored in directory `E`, but will be approximately 6.5K longer than that produced by a normal `LINK` command. Please note that the file *rtsys* must be present in the the current library directory. Further details about the `LINK` command and its qualifiers may be found in the C user guide on page 25.

Running the object code

The code produced by the Stand Alone Generator may now be executed as a machine code program:

```
*RUN e.welcome
```

The program will execute exactly as it would if executed by the Beebug C `RUN` command. You may like to arrange things such that the executable code is saved in the `$` directory as follows:

```
LINK/ STANDALONE/EXECUTABLE$ welcome welcome
```

or using abbreviations:

```
L/S/E=$ .welcome welcome
```

The code produced can be executed by typing:

```
*RUN welcome
```

If the library directory is set to $ (which it usually is), then executing the program can be simplified to:

```
*welcome
```

Setting the origin

The load and execute address of the executable code is set by default to OSHWM (PAGE), but may be set as required by using the optional qualifier `ORIGIN` (see user guide page 28). For example, to produce executable code that loads at &2500, enter:

```
LINK/STANDALONE/ORIGIN=&2 500 welcome
```

This feature is useful for producing code that will run on different systems which may have different PAGE settings. It may also be used to reserve space in the memory map for machine-code programs etc. In the example above, memory between PAGE and &2500 is free for other uses.

The `DEBUG` qualifier

The `NODEBUG` qualifier removes debugging information from object files before generating executable code. This has the effect of producing more compact code, at the expense of detailed error messages. In most programs, especially thoses containing a lot of functions, a fairly significant amount of memory can be saved. The qualifier is used as follows:

```
LINK/STANDALONE/NODEBUG welcome
```

or simply:

```
L/S/NOD welcome
```

Further details about the `DEBUG` qualifier may be found in the C user guide pages 22 and 28.

Passing arguments to C

A number of string arguments may be passed to the program by listing them after the name of the executable code. For example:

```
*RUN argtest argl arg2 arg3 ... argn
```

will pass the strings *`argl`*, *`arg2`*, *`arg3`* etc. to the C program `argtest`. As usual the arguments are passed to the array `argv`, and the number of arguments to the integer argc. An example showing how a C program interprets these arguments is given on page 30 of the C user guide.

Run-time qualifiers

All the usual qualifiers to the `RUN` command are available for the stand-alone code, except that they are identified by a slash character (/) followed by only a single letter. Thus, the qualifiers `/TRACEBACK`, `/INPUT`, `/OUTPUT` and `/ERROR` are specified by `/T`, `/I`, `/0` and `/E`. The following table summarises the run-time qualifiers.

optional qualifiers	default setting
`/[NO]T`	no traceback
`/INPUT`	input from keyboard
`/[NO]0`	output to VDU
`/[NO]E`	error stream to VDU

Effectively, a qualifier is a special case of a program parameter which is intercepted by the run-time system and is not assigned to the argv array. For example:

```
*utilityl /O=newfile red green blue
```

will execute the program `utilityl`, passing the arguments `red`, `blue` and `green`. The output stream is re-directed by the `/0` qualifier to the file `newfile`. Further details about the run-time qualifiers can be found in the C user guide, pages 31-34.

Error messages

There are two error messages produced by the stand-alone run-time system which are not present normally in Beebug C.

`Input file not found`
The file specified by the `/1` qualifier cannot be found.

`Bad qualifier`
A invalid qualifier has been found on the command line.

4. Maths Functions

Header file h.math

The Stand Alone Generator disc contains a new header file called h.math which declares the maths functions in the new library rtlib. Both these files should replace the earlier versions on your program disc. For completeness, the list below includes the constants `HUGE_VAL` and `PI`, which were in the original library and are also documented in the C user guide.

ceil

Type :function
Synopsis :
```
#include <h.math>
double ceil(double x);
```
Description :The `ceil` function returns the smallest integer not less than `x`, expressed as a double.

cos

Type :function
Synopsis :
```
#include <h.math>
double cos (double x);
```
Description :The `cos` function computes the cosine of `x` (measured in radians). A large magnitude argument may yield a result with little or no significance. The `cos` function returns the cosine value.

deg

Type :function
Synopsis :
```
#include <h.math>
double degidouble x);
```
Description :The `deg` function converts the floating-point number x from radian measure to degrees.

exp

Type :function
Synopsis :`#include <h.math>`
`double exp(double x);`
Description :The `exp` function computes the exponential function of x. This is the value of the mathematical constant *e* (2.718282) raised to the power of `x`. A range error error occurs if the value of `x` is too large (returns 0 if x is negative, and `HUGE_VAL` if positive).

fabs

Type :function
Synopsis :`#include <h.math>`
`double fabs(double x);`
Description :The `fabs` function computes the absolute value of the floating-point number `x`.

floor

Type :function
Synopsis :`#include <h.math>`
`double floor(double x);`
Description :The `floor` function computes the largest integer not greater than `x`.

fmod

Type function
Synopsis `:#include <h.math>`
`double fmod(double x, double y);`
Description :The fmod function computes the floating-point remainder of `x/y`. The result is a number with the same sign as `x`, and a magnitude less than `y`.

frexp

Type :function
Synopsis :
```
#include <h.math>
double frexp(double value, mt *exp);
```
Description :The `frexp` function breaks a floating-point number into a normalised fraction, and an integral power of 2. It stores the integer in the mt object pointed to by exp. It returns the value `x`, such that `x` is a double with magnitude in the range 0.5 to 1, or zero, and `value` equals x times 2 raised to the power `*exp`. If `value` is zero, both parts of the result are zero.

ftoi

Type :function
Synopsis :
```
#include <h.math>
int ftoi(double x);
```
Description :The `ftoi` function converts the floating-point number `x` to an integer.

HUGE_VAL

Type :macro
Synopsis :
```
#include <h.math>
#define HUGE_VAL l.7014118e38
```
Description :Expands to the positive double expression 1.7014118e38. This value is returned by maths functions if a range value occurs.

Ipow

Type :function
Synopsis :
```
#include <h.math>
double ipow(double x, int y);
```
Description :The `ipow` function computes `x` raised to the power `y`, where `x` is a floating-point number and `y` is an integer, This function is more efficient than the `pow` function, and should be used when `y` is an integer.

itof

Type :function
Synopsis :
```
#include <h.math>
double itof(int x);
```
Description :The `itof` function converts the integer `x` to floating-point representation.

ldexp

Type :function
Synopsis :
```
#include <h.math>
double ldexp(double x, int exp);
```
Description :The `ldexp` function multiplies the floating-point number x by the integral 2^{exp}. It returns the value of `x` times 2 raised to the power `exp`.

log

Type :function
Synopsis :
```
#include <h.math>
double log(double x);
```
Description :The `log` function computes the natural logarithm of `x`. A negative argument will result in the value `HUGE_VAL` being returned.

loglO

Type :function
Synopsis :
```
#include <h.math>
double loglO(double x);
```
Description :The `loglO` function computes the base-ten logarithm of `x`. A negative argument will result in the value `HUGE_VAL` being returned.

modf

Type :function
Synopsis :
```
#Include <h.math>
double modf (double value, double *lptr);
```
Description :The `modf` function breaks the argument value into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a double in the object pointed to by `iptr`. It returns the signed fractional part of `value`.

PI

Type macro
Synopsis :
```
#include <h.math>
#deflne PI 3.141593
```
Description :Expands to the value of PI.

pow

Type :function
Synopsis :
```
#Include <h.math>
double pow(double x, double y);
```
Description :The `pow` function computes x raised to the power of `y`, where both `x` and `y` are floating point numbers.

rad

Type :function
Synopsis :
```
#include <b.rnath>
double rad(double x);
```
Description The `rad` function converts the floating-point number `x` from degrees to radian measure.

sin

Type :function
Synopsis :
```
#include <h.math>
double sin(double x);
```
Description :The `sin` function computes the sine of `x` (measured in radians). A large magnitude argument may yield an inaccurate result.

sqrt

Type :function
Synopsis :
```
#include <h.math>
double sqrt(double x);
```
Description :The `sqrt` function computes the non-negative square root of `x` using the Newton-Ralphson method of approximations. Zero is returned if is negative.

tan

Type :function
Synopsis :
```
#include <h.math>
double tan(double x);
```
Description :The `tan` function computes the tangent of `x` (measured in radians). A large magnitude argument may yield an inaccurate result. A range error occurs if the value is not computable (e.g. x = P1/2 radians). In this case the value of the macro `HUGE_VAL` is returned.

Example program

The following program illustrates the use of two of the above functions by drawing a sine wave on the screen. The source code for this program is supplied on disc, and is called `c. sinwave.`

```
/* Beebug C Sine Wave */

#include <h.math>
#include <h.stdlib>

void main (void)
{
     int i;

     mode (4);
     for (i = 0; ± < 1280; i++)
          plot (69, 1, sin(rad(i) ) * 500 + 500);
}
```

To produce stand-alone code for this program type:

```
COMPILE sinwave
LINK/S/NOD/E=$ . sinwave sinwave
```

The program can now be run using:

```
*s inwave
```

This assumes that the current directory is **$**.

Note that the above from relies on the function prototype facilities of Beebug C, to enable conversions from `int` to `float` for the `rad` function, and from `float` to `int` for the `plot` function. For this reason, it will not work on other C systems which do not support the ANSI/ISO function prototype extension.

Appendix A

Summary of Maths Functions

Listed below is an alphabetical list of all the maths functions and macros available in Beebug C. Each function or macro is listed together with its file type, the header file which declares it, and a brief description.

Function	Type	Header	Description	Page
ceil	f	h.math	return the smallest integer	9
cos	f	h.math	return the cosine value	9
deg	f	h.math	convert from radian measure to degrees	9
exp	f	h.math	compute the exponential function	10
fabs	f	h.math	compute absolute value of FP number	10
floor	f	h.math	return the largest integer	10
fmod	f	h.math	compute the floating-point remainder	10
frexp	f	h.math	break number into fraction and power	11
ftoi	f	h.math	convert FP number to an integer	11
HUGE_VAL	rn	h.math	the maximum positive double	11
ipow	f	h.math	compute an integer power	11
itof	f	h.math	convert integer to floating-point	12
ldexp	f	h.math	multiply FP number by 2exp	12
log	f	h.math	compute the matural logarithm	12
log10	f	h.math	compute base-lO logarithm	12
modf	f	h.math	break riumber into integer and fraction	13
PI	rn	h.math	constant PI (3.141593)	13
pow	f	h.math	raise FP number to FP power	13
rad	f	h.math	convert from degree measure to radians	13
sin	/	h.math	compute the sine value	14
sqrt	f	h.math	compute the square root value	14
tan	f	h.math	compute the tangent value	14