mophun^{TM} Resource Compiler

Contents

Chapter 1

Introduction

Applications consist of code and data. The code make up the instructions that perform the actions of an application, data is used by the application to perform it's actions and present information to the user. For example it may be graphical bitmaps, sounds or text. MORC is a tool that collects and prepares data for use in a mophun application.

MORC reads a description of the data from a file and generates data in a format that is usable on the mophun platform. The user creates a resource description file (.rc) which lists resources that will be used by the program and processes it with MORC which generates a binary output file that can be linked into the application, it also creates a C header (.h) file which defines additional information about each resource, such as the size of resources and other information needed by the user of the resource.

Chapter 2

Invocation

This chapter describes how to invoke MORC from the command line or in scripts.

2.1 Options

```
morc [options] INPUT [OUTPUT]
-as COMMAND     Set the assembler command used when
                compiling the resources.
-AOPTION        Pass OPTION to the assembler.
-cpp COMMAND    Set the preprocessor command.
-data           Output a data file instead of an object file.
-DNAME[=VAL]    Define a preprocessor define.
-h HEADER       Set the name of the output header file.
-IPATH          Add PATH to list of directories
                searched for files.
-nocpp          Do not run the preprocessor
-o OBJECTFILE   Set the name of the output object file.
-save-temps     Do not remove temporary files (.s files)
-v              Verbose output.
```

If no output or header filename is specified, MORC uses the name of the input file but changes the to filename extension to .o and .h respectively.

Chapter 3

Language

This chapter describes the syntax of the MORC resource definition language.

3.1 Syntax

The general MORC syntax consists of directives followed by a number arguments. Syntax is case-sensitive and all directives and attributes are specified in upper-case.

3.1.1 Basic syntax

- Filenames are specified as strings within double quotes, for example: “splash-screen.bmp”.
- Numbers are specified using *C* style syntax. A *0x* prefix indicates a number in hexadecimal notation, a *0* prefix indicates a number in octal notation.
- Numeric expressions may be used anywhere a number is expected. All *C* arithmetic operators are supported including parentheses.
- A list of arguments or resource attributes are normally specified as a list of white-space delimited tokens.
- A pair of numbers may be specified as two consecutive numbers optionally separated by a colon. The pair may also be surrounded by parentheses, for example the pair 8,8 may be specified as *“8 8”, “8,8”,(8 8)* or *(8,8)*.
- Strings are surrounded by double quotes. All ANSI C escape codes are supported except *’\”’*. To insert arbitrary values or characters in a string prefix the character by “\x” followed by a hexadecimal value or specify “\o” followed by an octal value. Example: “\x22” specifies the character code for a double quote.

3.1.2 Preprocessor

To make the system more flexible MORC preprocesses the input file before parsing it. The syntax is the same as the normal *C* preprocessor.[1] This allows you to conditionally include or exclude parts of the file, create macros and define values for replacement.

3.1.3 Bitmap formats

Syntax: [FORMAT *FORMAT_SPECIFIER*] [OFFSET *NUM-EXPR]* [SIZE *WIDTH HEIGHT*] [CEN-TER *X Y*] [PALETTE “FILENAME”]

The mophun platform supports several different bitmap graphic formats. A bitmap has several attributes, including bit depth, dimensions and palettes.

- *FORMAT* specifies the color format.
- *OFFSET* specifies the offset in the palette used by this bitmap.
- *SIZE* specifies the size of the bitmap.
- *CENTER* specifies the center of the bitmap (used by *SPRITE* objects).
- *PALETTE* specifies the palette used by the file.

The following color format specifiers are supported:

- *IND2* indexed monochrome.
- *IND4* indexed 2 bits per pixel
- *IND16* indexed 4 bits per pixel.
- *IND256* index 8 bits per pixel.
- *RGB332* direct 8 bits per pixel, 3 bits for red, 3 bits for green and 2 bits for blue.
- *RGB555* 15 bits per pixel stored in a 16 bit number.
- *HICOLOR/RGB565* 16 bits per pixel.
- *TRUECOLOR* 24 bits per pixel.
- Bits per pixel specified as a number.

The following bitmap file types are supported:

- Windows bitmap files (.bmp). RLE decoding is not supported.
- Raw bitmap files (.raw).

[1] In fact, it *is* the *C* preprocessor.

  - Lines are padded to the nearest whole byte.
  - For pixel depths less than 8 bits per pixel are stored with leftmost pixels in the least significant bits if a mophun image format is specified otherwise pixels are stored with leftmost pixels in the most significant bits.
  - 24 bit image formats are stored in RGB byte order.
  - 15/16 bit image formats are stored in RGB order with red bits in the most significant bits etc.

- Mophun sprite files (.spr), these are just mophun raw image data with a SPRITE header.

3.1.4 Packing resources

`Syntax: id PACKED [OFFSET COUNT] TYPE ...`

If the *PACKED* resource attribute is specified the data is compressed before output, MORC will also output two special defines in the output header for packed resources. It is possible to specify the compressor settings by specifying *OFFSET* and *COUNT*, however, this is not recommended since by default the compressor tries to find the best compressor settings.

- *OFFSET* specifies the size in bits of the compressor sliding window.
- *COUNT* specifies the maximum length in bits used for matching redundant data in the compressed output.

Defines:

- *UNPACK_SIZE* specifies the size of the unpacked resource data.
- *PACKED* is defined if a resource is packed.

3.2 Resource directives

To define a resource the following syntax is used:

`id [PACKED [CNT OFS]] COMMAND ARGUMENTS`

This will create a resource called *id* of the type specified by *COMMAND* and with the attributes specified by *ARGUMENTS*. If *PACKED* is specified the resource is compressed on output. In the following sections the all resource directives are described. Some resources define extra information in the output header, for example a *FONT*[2] resource defines the width, height and bits per pixel used by a bitmap font:

[2] See Section 3.2.3

```
MYFONT FONT 6 8 FORMAT IND2 “myfont.bmp” “ABCDE-
FGHIJKLMNOPQRSTUVWXYZ”
```

The example above will generate the following defines in the output header which may then be used to initialize a *VMGPFONT* structure:

```
#define MYFONT_WIDTH     6
#define MYFONT_HEIGHT    8
#define MYFONT_BPP       1
#define MYFONT_PALINDEX  0
#define MYFONT_CHARSIZE  6
#define MYFONT_CHARCOUNT 26
#define MYFONT_CHARTBLSIZE 90
```

The possible defines for each resource type is described in the “Defines” section of each resource.

3.2.1 BEEP

Syntax: `id BEEP [DEFVOL] { FREQ, DUR [: VOL]... }`
Syntax: `id BEEP [DEFVOL] “FILENAME”`

The *BEEP* directive generates a sequence of beeps.

- *DEFVOL* specifies the default volume for subsequent beeps
- *FREQ* specifies the sixteen bit freqeuency
- *DUR* specifies the sixteen bit duration in milli-seconds
- *VOL* specifies the optional volume of the beep in the range 0-255, where 255 is maximum volume.
- *FILENAME* specifies a file with beeps in raw beep format:
  Frequency: 16 bit
  Duration: 16 bit
  Volume: 8 bit

Defines:

- *COUNT:* the number of beeps in the sequence.

Example:

`beep_r.rc`:

```
SECTION DATA

bleep BEEP 128
{
  1792 32 : 255
  1920 32
  2048 32
  2176 32
  2304 32
}
```

beep.c:

```
#include <vmgp.h>
#include "beep_r.h"

main ()
{
  vPlayResource(&bleep, bleep_SIZE, SOUND_TYPE_BEEP|SOUND_FLAG_LOOP);
  while ((vGetButtonData() & KEY_FIRE) == 0);
}
```

3.2.2 DATA

Syntax: [id] DATA “filename”
Syntax: [id] DATA { ... }

The *DATA* directive creates a resource for arbitrary data, it does not assume anything about it’s contents. The first version of the directive simply inserts the contents of the specified file.

The second version defines the data inline in the resource file, each element in the data list consists of one or more of the following separated by a colon:

- A numeric expression (see 3.1.1 on page 5), this is treated as byte value if it fits within a byte, otherwise as a 16 bit value if it fits otherwise as 32 bits.
- A numeric constant is a special numeric value with a size suffix. The following suffixes are supported: U and L indicate a 32 bit value, S or H indicates a 16 bit value.
- A character withing single quotes.
- A string withing double quotes.
- A *FILL* or *PAD* directive, see Section 3.3.3 for more information.
- An *ALIGN* directive, see Section 3.3.1 for more information.

Example:

data_r.rc:

```
data DATA
{
  0,1,2,3,      // Byte Numbers
  4S,5H,        // 16 bit values
  8L,12U,       // 32 bit values
  "STRING",     // A string
  'C',          // A character
  ALIGN 64,     // Align to 64 bytes
  PAD 64, 0xff  // Pad data
}

SECTION DATA
tile DATA "tile.raw"
```

data.c:

```
#include <vmgp.h>
#include <vstream.h>
#include "data_r.h"

unsigned char buf[data_SIZE];

main ()
{
  int fd = vResOpen (NULL, data);
  vResRead (fd, buf, data_SIZE);
  vResClose (fd);

  vDrawTile (tile, VCAPS_RGB332, 0, 0);
  vFlipScreen (1);
}
```

3.2.3 FONT

Syntax: `id FONT [NOCASE] FONTSIZE [GFX_ATTR] “FILENAME” “CHARSET”`

The *FONT* directive creates a mophun font.

- *NOCASE* indicates that a case-independent character table should be generated.
- *FONTSIZE* specifies the width and height of the font.
- *GFX_ATTR* specifies the bitmap format of the font, see Section 3.1.3.

- *FILENAME* specifies the font bitmap.
- *CHARSET* specifies the characters in the font in the order they appear in the bitmap.

In the input bitmap each character is stored in left to right and top to bottom order.

Defines:

- *GLYPHCOUNT*: the number of bitmaps in the font.
- *GLYPHSIZE:* the size of a single font bitmap in bytes.
- *BPP*: bits per pixel in the font glyph.
- *WIDTH*: the width of the font in pixels.
- *HEIGHT*: the height of the font in pixels.
- *PALINDEX*: the palette index used by the font.
- *CHARTBLSIZE:* the size of the character table. Add this to the beginning of the font data to get the first font bitmap data.

Example:

`font_r.rc:`

```
SECTION DATA
font FONT 6 8 FORMAT IND2
     "font.bmp"
     "1234567890ABCDEFGHIJKLMNOPQRSTUVXYZ+-*/(),._=:<>!?\x22% "
```

`font.c:`

```
#include <vmgp.h>
#include "font_r.h"

VMGPFONT theFont =
{
  font + font_CHARTBLSIZE,
  font,
  font_BPP,
  font_WIDTH,
  font_HEIGHT,
  font_PALINDEX
};


main ()
```

```
{
  vSetBackColor (VMGP_BLACK);
  vSetForeColor (VMGP_WHITE);
  vSetActiveFont (&theFont);

  vPrint (MODE_BLOCK, 0, 0, "1234567890");
  vPrint (MODE_BLOCK, 0, font_HEIGHT + 1, "ABCDEFGHIJK");
  vPrint (MODE_BLOCK, 0, font_HEIGHT * 2 + 1, "LMNOPQRSTUVXYZ");
  vPrint (MODE_BLOCK, 0, font_HEIGHT * 3 + 1, "+-*/(),._=:<>!?\"% ");
  vFlipScreen (1);
}
```

3.2.4 GROUP

Syntax: `id GROUP { [resource directives] }`

The group directive allows you to group several resources together into a single resource. Within scope of the group you can use all resource directives and control directives except the SECTION 3.3.7 and SYMBOL 3.3.8 directives.

Defines:

- *COUNT:* the number of subresources in the group.
- *SIZE:* the size of a the group (all resources).

In addition all the normal resource defines are generated. The *OFS* value for a resource defines the offset within the group.

3.2.5 METAINFO

Syntax: `[id] METAINFO [CERTSIZE] { KEY : VALUE... }`

The *METAINFO* directive constructs a mophun meta info list. A meta info list consist of:

- A certificate (optional), there is no way to create the actual certificate that is done by the application certification process. To create a meta info list without a certificate specify 0 for *CERTSIZE*.
- String key, value pairs. Used for descriptive purposes or for storing information about the application.
- Application icons. Icons are stored in 8 bit RGB332 format. The maximum size is 255x255 pixels.

Example:

metainfo_r.rc:

```
METAINFO
{
  "Title" : "Space Blaster"
  "Vendor" : "Joe Coder"
  "Copyright" : "Copyright (C) Joe Coder 2002"
  ICON : "16x16.bmp"
  ICON "Another icon" : "16x16.bmp"
}
```

metainfo.c:

```
main ()
{

}
```

3.2.6 PALETTE

Syntax: `id PALETTE [COUNT] “FILENAME”`

The *PALETTE* directive creates a mophun palette from an input file. MORC supports palettes in JASC Paint Shop Pro format, MS RIFF palettes and raw palettes in RGB format. It is also possible to specify an image and use the palette stored in the image file. If *COUNT* is specified, a maximum of *COUNT* palette entries are used.

Defines:

- *COUNT*: the number of entries in the palette.

Example:

palette_r.rc:

```
SECTION DATA

pal PALETTE "16x16.bmp"
spr SPRITE "16x16.bmp"
```

palette.c:

```
#include <vmgp.h>
#include "palette_r.h"

main ()
{
```

```
  vSetPalette (pal, 0, pal_COUNT);
  vDrawObject (0, 0, &spr);
  vFlipScreen (1);
}
```

3.2.7 SOUND

Syntax: `id SOUND [PRIO N] [LOOPSTART N] [LOOPEND N] [PCM|ADPCM] “FILENAME”`

The *SOUND* directetive takes a windows *.wav* file as input and produces a sound resource in mophun format. The input file must be in 8- or 16-bit mono/stereo PCM format. ADPCM encoding requires the file to be in 16-bit mono PCM format.

- *PRIO*: The priority of the sound, a value between 0-127, default is 64.
- *LOOPSTART*: the sample-frame of the start loop-point. Default is 0.
- *LOOPEND*: the sample-frame of the end loop-point. Default is the last sample-frame.
- *PCM*: store the sound data in linear PCM format.
- *ADPCM*: store the sound data in ADPCM encoded format.

Example:

sound_r.rc:

```
SECTION DATA

CHORD SOUND "chord.wav"
DRUMLOOP SOUND LOOPSTART 320 "drum.wav"
```

sound.c:

```
#include <vmgp.h>
#include <vsound.h>
#include <vmgputil.h>

#include "sound_r.h"

int main()
{
  HSOUND hChord, hDrum;
  vSoundInit();

  hChord = vSoundGetHandle((int32_t*)CHORD);
  hDrum = vSoundGetHandle((int32_t*)DRUMLOOP);
```

```
  vSoundPlay(hChord, 0);
  vSoundPlay(hDrum, SNDPLAY_LOOPING);

  while (vGetButtonData() == 0)
    ;

  vSoundDispose();
  return 0;
}
```

3.2.8 SPRITE

Syntax: `id SPRITE [GFX_ATTR] “FILENAME”`

The *SPRITE* directive creates a mophun sprite object from an image. If no image attributes are specified the sprite uses the information from the input file, see Section 3.1.3 for more information on bitmap attributes.

Defines:

- *WIDTH*: the width of the sprite.
- *HEIGHT*: the height of the sprite.
- *PALINDEX:* the palette indexed used by the sprite.
- *FORMAT*: the color format of the sprite.
- *CENTERX*: the horizontal center-point of the sprite.
- *CENTERY*: the vertical center-point of the sprite.

Example:

sprite_r.rc:

```
SECTION DATA

pal PALETTE "truck.bmp"

spr0 SPRITE FORMAT RGB332 CENTER (8,8) "16x16.bmp"
spr1 SPRITE "truck.bmp"
```

sprite.c:

```
#include <vmgp.h>
#include "sprite_r.h"
```

```
main ()
{
  vSetPalette (pal, 0, pal_COUNT);
  vDrawObject (0, 0, &spr0);
  vDrawObject (spr0_WIDTH, 0, &spr0);
  vDrawObject (0, spr0_HEIGHT, &spr0);
  vDrawObject (spr0_WIDTH, spr0_HEIGHT, &spr0);

  vDrawObject (0, 50, &spr1);
  vFlipScreen (1);
}
```

3.2.9 STRING

Syntax: id STRING “VALUE”
Syntax: id STRING { ... }

If *VALUE* is specified the *STRING* directive defines a null-terminated string. Otherwise it defines a string of characters, see Section 3.2.2 for information on the syntax.

Example:

string_r.rc:

```
#ifndef LANG
#define LANG EN
#endif

SECTION DATA

STR_TITLE STRING "Space Blaster"

#if LANG == EN
STR_NEWGAME STRING "New Game"
STR_QUIT STRING "Quit"
#elif LANG == SE
STR_NEWGAME STRING "Nytt Spel"
STR_QUIT STRING "Avsluta"
#endif
```

string.c:

```
#include <vmgp.h>
#include "string_r.h"

const char *strings[] =
{
```

```
  STR_TITLE,
  STR_NEWGAME,
  STR_QUIT,
  0
};

main ()
{
  int i;
  int y;

  VIDEOCAPS vc;

  vc.size = sizeof (vc);
  vGetCaps (CAPS_VIDEO, &vc);

  vSelectFont (FONT_SIZE_NORMAL,
               FONT_STYLE_NORMAL | FONT_EFFECT_SHADOW_LOWERRIGHT,
               0);

  vClearScreen (VMGP_BLUE);
  vSetForeColor (VMGP_RED);
  vSetBackColor (0);
  vSetTransferMode (MODE_TRANS);

  y = 0;
  for (i = 0; strings[i]; i++)
  {
    const char *str = strings[i];
    int extent = vTextExtent (str);
    int x = (vc.width / 2) - (extent & 0xffff) / 2;
    vTextOut (x, y, str);
    y += extent >> 16;
  }

  vFlipScreen (1);
  while (!(vGetButtonData () & KEY_FIRE));
}
```

3.2.10 TEXT

Syntax: id TEXT “FILENAME”
Syntax: id TEXT { ... }

The *TEXT* directive is similar to *STRING* but takes a *FILENAME* as parameter instead of a literal string.

3.2.11 TEXTURE

Syntax: `id TEXTURE [GFX_ATTR] “FILENAME”`

The TEXTURE directive generates texture data from an image. If no image attributes are specified the texture uses the information from the input file, see Section 3.1.3 for more information on bitmap attributes.

Defines:

- *WIDTH*: the width of the texture.
- *HEIGHT:* the height of the texture.
- *FORMAT*: the color format of the texture.
- *LODS:* the x and y lod values.

Example:

3.2.12 TILESET

Syntax: `id TILESET [WIDTH HEIGHT] [GFX_ATTR] “FILENAME”`

The *TILESET* directive creates a tileset from an input image. *WIDTH* and *HEIGHT* specify the size of a single tile[3], *GFX_ATTR* specifies the bitmap attributes of the tile (see Section 3.1.3 for more information on bitmap attributes).

Defines:

- *COUNT*: the number of tiles in the tileset.
- *TILESIZE*: the size of a single tile in bytes.
- *WIDTH*: the width of a tile in pixels.
- *HEIGHT:* the height of a tile in pixels.
- *TILEFMT*: the tile format, suitable for passing to the *vDrawTile* function.

[3]Note that mophun currently only supports 8x8 tiles

Example:

tileset_r.rc:

```
SECTION DATA

pal PALETTE "font8x8.bmp"
tileset TILESET "font8x8.bmp"
```

tileset.c:

```
#include <vmgp.h>
#include "tileset_r.h"

main ()
{
  VIDEOCAPS vc;
  int x = 0;
  int y = 0;
  int n = 0;
  char *tile = tileset;

  vc.size = sizeof (vc);
  vGetCaps (CAPS_VIDEO, &vc);

  vSetPalette (pal, 0, pal_COUNT);

  for (n = 0; n < tileset_COUNT; n++)
  {
    vDrawTile (tile, tileset_TILEFMT, x, y);

    x += tileset_WIDTH;
    if (x >= vc.width)
    {
      x = 0;
      y += tileset_HEIGHT;
    }

    tile += tileset_TILESIZE;
  }

  vFlipScreen (1);
}
```

3.3 Control directives

There are also a number of control directives that affect subsequently defined resources. Control directives are entered as:

```
COMMAND ARGUMENTS
```

3.3.1 ALIGN

Syntax: `ALIGN COUNT`

Align the following resource to a boundary of *COUNT* bytes. Note that many resources have an implicit alignment, so this directive is really only necessary when defining resource with the *DATA* directive.

Example:

```
ALIGN 4     // Align following resource to four bytes
mydata DATA “data.dat”
```

3.3.2 DEFINE

Syntax: `DEFINE id “VALUE”`

Create a *C* define in the output header. The value of the output define is not quoted.

Example:

```
DEFINE MAGIC “The Value”
DEFINE MAGIC2 0x12345678
```

3.3.3 FILL

Syntax: `FILL COUNT, VAL`

Insert *COUNT* bytes initalized to *VAL* at the current location in output.

Example:

```
mydata DATA “data.dat”
FILL 1,0    // Terminate data
```

3.3.4 GROUP

See Section 3.2.4.

3.3.5 INCLUDE

Syntax: `INCLUDE “FILENAME”`

Include *FILENAME* from the output header file.

Example:

```
INCLUDE “mydefs.h” // This will be included in
                   // the output header file
```

3.3.6 PAD

Syntax: `PAD COUNT, VAL`

The *PAD* directive is equivalent to the *FILL* directive and may be used anywhere a *FILL* directive is expected.

3.3.7 SECTION

Syntax: `SECTION DATA`
Syntax: `SECTION “NAME”`
Syntax: `SECTION LIST “NAME”`

The *SECTION* directive changes the output section of subsequently defined resources. A resource file can have multiple *SECTION* statements. By default MORC writes resources to the “resource” section, resources in this section may be accessed with the *vResOpen* API function.[4] If *NAME* is equal to *“DEFAULT”* MORC switches to the resource section.

The *SECTION DATA* directive puts resources in the normal data section, this makes it possible to use the resources as ordinary variables from a C/C++ program.

It is also possible to put the resources in a specific section if you specify a section name.

If *LIST* is specified it specifies the name of resource list section, see Section 3.3.8 for more information on resource lists.

Example:

section_r.rc:

```
// This goes into the resource section
// and is readable with the stream API

tiledata DATA "tile.raw"

// Change to the data section
```

[4] See the Mophun API Reference Guide

```
// The following resources are normal variables
SECTION DATA
string STRING "Hello, world!"

sprite SPRITE FORMAT RGB332 "truck.bmp"

// Change back to the resource section
SECTION "DEFAULT"
SAVESTATE DATA
{
  FILL 256,0
}

// Change to my special section
SECTION ".frogger"
SYMBOL "Frogger"
DATA
{
  "This is full of highly specialized ",
  'd','a','t','a',0,
  ALIGN 64
}
```

section.c:

```
#include <vmgp.h>
#include <vstream.h>
#include "section_r.h"

char tile[tiledata_SIZE];

main ()
{
  // Read resource from resource section
  int fd = vResOpen (NULL, tiledata);
  vResRead (fd, tile, tiledata_SIZE);
  vResClose (fd);

  vDrawTile (tile, VCAPS_RGB332, 0, 0);

  // Use variable stored in the data section
  vTextOut (0,8,string);


  // Open a resource for writing
  fd = vResOpenMode (NULL, SAVESTATE, STREAM_WRITE);
```

```
    // Write our special section resource to the resource section
    vResWrite (fd, Frogger, sizeof (Frogger));
    vResClose (fd);

    vFlipScreen (1);
}
```

3.3.8 SYMBOL

Syntax: SYMBOL “NAME”
Syntax: SYMBOL LIST “NAME”

The *SYMBOL* directive defines the name of the output data symbol. An output symbol is a name of a data array that contains all the resources in a section. This makes it possible to group several resources together in a chunk of data and access individual resources data by adding an offset to the beginning of the data symbol.

If *LIST* is specified the *SYMBOL* directive defines the name of the resource list symbol. The list symbol is the name of an array of 32 bit offsets into the resource data symbol for the resources in a section. The first resource has offset of 0 in the first entry of the resource list.

Example:

See Section 3.3.7 for an example of resource symbols.