RIC File Format

RIC stands for Robot P**ic**ture or Graph**ic**. This file format is used by the NXT to store and draw pictures. The format isn't a bitmap it's more like a program. The file consists of a series of commands. Since all RIC files brought by LEGO consist only of three commands I will only explain them here.

General command structure

A command should always have a even amount of bytes. Each command starts with a word that represents the length of the following data, that is the command size minus 2. The second word represents the command number (opcode). After that the command parameters follow.

```
struct {
    uint16_t len;
    uint16_t opcode;
    ...
};
```

Description Command

Opcode: 0

This command has a fixed size of 10 bytes. As described in the last paragraph the first two words are command length and opcode. For displaying a picture on NXT this command is useless it's only needed for displaying in NXT-G. It contains options, width and height. All this parameters are words. The options parameter is unused.

```
struct ric_description {
    uint16_t len;
    uint16_t opcode; // 0
    uint16_t options; // 0
    uint16_t width;
    uint16_t height;
};
```

Sprite Command

Opcode: 1

The sprite command is the command that holds the bitmap. It consists of the words 'dataaddr', 'rows', 'rowbytes' and the actual bitmap data. The field 'dataaddr' is commonly set to 1. The field 'rows' is nothing else then the field 'height' from the last paragraph. The field 'rowbytes' represents the number of bytes used by one row. Since 8 pixels fit in one byte this is 'width'/8 rounded up.

by Janosch Gräf <janosch.graef@gmx.net>

The bitmap has a size of 'rowbytes'*'rows'. So in each row are 'rowbytes' bytes. These bytes are filled with the pixels. Each byte can hold up to 8 pixels. The last bytes doesn't have to hold 8 pixel since the width doesn't have to be dividable through 8. The first pixel in a byte is the most significant byte. This is a row with 11 pixels:

Byte	1								2							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Pixel	0	1	2	3	4	5	6	7	8	9	10	X	X	X	X	X

```
struct ric_sprite {
     uint16_t len;
     uint16_t opcode;
     uint16_t dataaddr;
     uint16_t rows;
     uint16_t rowbytes;
     uint8_t data[X];
};
```

CopyBits Command

Opcode: 3

The CopyBits command has a fixed length of 20 bytes and is used to display the sprite. It consists of the fields 'options', 'dataaddr', 'src' and 'dest'. The field 'options' specifies the operation that is done. Options are 'Copy', 'CopyNot', 'Or' and 'BitClear'. We do only use 'Copy'. 'dataaddr' is the same as in last paragraph. It must have the same value. 'src' specifies the source rectangle (for structure see below) for the shown bitmap. Usually it has the position (0|0) and size ('width'|'height'). 'dest' specifies the point where the bitmap is displayed. Since the running program can put pictures on the display wherever it want, this is only a offset and is usually (0|0).

```
struct pic_point {
     int16_t x;
     int16_t y;
};

struct pic_rect {
     struct pic_point point;
     int16_t width;
     int16_t height;
};

struct pic_copybits {
```

by Janosch Gräf <janosch.graef@gmx.net>

```
    uint16_t len; // 20
    uint16_t opcode; // 3
    enum {
        RIC_COPY = 0,
        RIC_COPYNOT = 1,
        RIC_OR = 2,
        RIC_BITCLEAR = 3
    } options:16;
    uint16_t dataaddr;
    struct ric_rect src;
    struct ric_point dest;
};
```