Cairo in Perl 6 - NativeCall bindings

David Warring July 2017

- `libcairo` is a 2D graphics native library
- Working on Perl 6 Cairo module
- Simple 'Hello world' example
- In 3 slides of code
- Natve C Structs
- Callbacks
- Workplace?

Perl 6 Native Types Recap

Perl 6 has PHP/Ruby/Python like dynamic arrays:

```
my @a = (42, "Hi", [3,4,5]);
say @a[1];
```

But also C/C#/Java like compact native arrays and variables:

```
my uint32 @b = 123, 456, 789;
my uint8 $bytes;
```

Design goal: Native arrays faster in tight loops and critical code.

Optimisation is still a work in progress (JIT).

Perl 6 has an affinity to native libraries.

Cairo - hello world in C

```c
#include <cairo.h>

int
main (int argc, char *argv[])
{
    cairo_surface_t *image
        = cairo_image_surface_create(
                CAIRO_FORMAT_ARGB32, 240, 80
            );
    cairo_t *ctx = cairo_create ();

    cairo_move_to (ctx, 10.0, 50.0);
    cairo_show_text (ctx, "Hello, world");

    cairo_destroy (ctx);
    cairo_surface_write_to_png (image, "hello.png");
    cairo_surface_destroy (image);
    return 0;
}
```

Definitions from /usr/include/cairo/cairo.h

```
typedef struct _cairo cairo_t;
typedef struct _cairo_surface cairo_surface_t;

cairo_public cairo_status_t
cairo_surface_write_to_png (cairo_surface_t *surface,
                            const char *filename);
typedef enum _cairo_format {
    CAIRO_FORMAT_INVALID   = -1,
    CAIRO_FORMAT_ARGB32    = 0,
    CAIRO_FORMAT_RGB24     = 1,
    /* ... */
} cairo_format_t;

cairo_public cairo_surface_t *
cairo_image_surface_create (cairo_format_t format,
                            int width, int height);

cairo_public void
cairo_move_to (cairo_t *cr, double x, double y);

cairo_public void
cairo_show_text (cairo_t *cr, const char *utf8);
```

Hello World in Perl 6

```
use Cairo;

my Cairo::Image $img .= create( Cairo::FORMAT_ARGB32,
                                128, 128);
my Cairo::Context $ctx .= new($img);

$ctx.move_to(10, 50);
$ctx.show_text: "Hello world";

$img.write_png: "hello.png"
```

'Cairo' Module Skeleton

```
unit Class Cairo;

class cairo_surface_t is repr('CPointer') {
    # native calls
}

class Surface {
    # user methods
}

class Image is Surface {
    # user methods
}

class cairo_t is repr('CPointer') {
    # native calls
}

Context {
    # external methods
}
```

NativeCall Subroutines and Methods

```
class cairo_surface_t is repr('CPointer') {}
sub cairo_surface_write_to_png(cairo_surface_t surface,
                               Str $filename)
     returns int32
     is native($cairolib)
     {*}
```

Is better written as:

```
class cairo_surface_t is repr('CPointer') {
    method write_to_png(Str $filename)
        returns int32
        is native($cairolib)
        is symbol('cairo_surface_write_to_png')
        {*}
}
```

Then called as:

```
    $surface.write_to_png("myfile.png");
```

Cairo Page1: bindings, enums, surface

```
unit module Cairo;
my $cairolib;
BEGIN {
    $cairolib = ('cairo', v2);
}
use NativeCall;

our enum Format (
     FORMAT_INVALID => -1,
    "FORMAT_ARGB32"   ,
    "FORMAT_RGB24"    ,
    # ...
);

class cairo_surface_t is repr('CPointer') {
    method write_to_png(Str $filename)
        returns int32
        is native($cairolib)
        is symbol('cairo_surface_write_to_png')
        {*}
}
```

Cairo Page 2: Surface and Image classes

```
class Surface {
    has cairo_surface_t $.surface
        handles <write_to_png>;
}

class Image is Surface {
    sub cairo_image_surface_create(int32 $format,
                                   int32 $width, int32 $height)
        returns cairo_surface_t
        is native($cairolib) {*}

     method create(Int(Format) $format,
                   Int(Cool) $width,
                   Int(Cool) $height) {
        my $surface = cairo_image_surface_create(
             $format, $width, $height)
        return self.new: :$surface;
    }
}
```

Cairo Page 3: Context class

```
class cairo_t is repr('CPointer') {
    method show_text(Str $utf8)
        is native($cairolib)
        is symbol('cairo_show_text') {*}
    method move_to(num64 $x, num64 $y)
        is native($cairolib)
        is symbol('cairo_move_to' {*}
}
class Context {
    sub cairo_create(cairo_surface_t $surface)
        returns cairo_t
        is native($cairolib)
        {*}

    has cairo_t $.context handles <show_text move_to>;

    method new(Surface $surface) {
        my $context = cairo_create($surface.surface);
        self.bless(:$context);
    }
 }
```

The 'Hello world' program again:

```
use Cairo;

my Cairo::Image $img .= create( Cairo::FORMAT_ARGB32,
                                128, 128);
my Cairo::Context $ctx .= new($img);

$ctx.move_to(10, 50);
$ctx.show_text: "Hello world";

$img.write_png: "hello.png"
```

C Structs in NativeCall

These can be directly declared. For example::

```
typedef struct _cairo_matrix {
    double xx; double yx;
    double xy; double yy;
    double x0; double y0;
} cairo_matrix_t;
cairo_public void
cairo_matrix_translate (cairo_matrix_t *matrix,
                        double tx, double ty);
```

Is declared as:

```
our class cairo_matrix_t is repr('CStruct') {
    has num64 $.xx; has num64 $.yx;
    has num64 $.xy; has num64 $.yy;
    has num64 $.x0; has num64 $.y0;

    method translate(num64 $tx, num64 $ty)
        is native($cairolib)
        is symbol('cairo_matrix_translate')
        {*}
```

Matrix wrapper class:

```
class Matrix {
    has cairo_matrix_t $.matrix handles <
        xx yx xy yy x0 y0
    > .= new: :xx(1e0), :yy(1e0);

    method translate(Num(Cool) $sx, Num(Cool) $sy) {
        $!matrix.translate($sx, $sy);
        self;
    }
}
```

Test Program:

```
use Cairo;

my $matrix = Cairo::Matrix.new;
say $matrix;
# xx => 1, yx => 0, xy => 0, yy => 1, x0 => 0, y0 => 0)
$matrix.translate(10,20);
say $matrix;
# xx => 1, yx => 0, xy => 0, yy => 1, x0 => 10, y0 => 20)
```

Callbacks

NativeCall is bidirectional. We can call Perl 6 routines from C code.

```
cairo_public cairo_status_t
cairo_surface_write_to_png_stream (
            cairo_surface_t *surface,
            cairo_write_func_t write_func,
            void *closure);
```

In Perl 6:

```
our class cairo_surface_t is repr('CPointer') {
# ...
        method write_to_png_stream(
            &write-func(
                StreamClosure, Pointer[uint8],
                uint32 --> int32),
            StreamClosure)
         returns int32
         is native($cairolib)
         is symbol('cairo_surface_write_to_png_stream')
         {*}
}
```

Surface 'Blob' method

Write a Cairo surface to an in-memory buffer.

```
class Surface {
# ...
    method Blob(UInt :$size = 16_000 --> Blob) {
        my $buf = CArray[uint8].new;
        $buf[$size] = 0;
        my $closure = StreamClosure.new: :$buf,
                  :buf-len(0), :n-read(0), :$size;
        $!surface.write_to_png_stream(
                              &StreamClosure::write,
                              $closure);
        return Blob.new: $buf[0 ..^ $closure.buf-len];
    }
}
```

The StreamClosure class

Call-back class. Used to read and write buffers

```
class StreamClosure is repr('CStruct') is rw {
    has CArray[uint8] $!buf;
    has size_t $.buf-len;
    has size_t $.size;
    method TWEAK(CArray :$buf!) { $!buf := $buf }
    method buf-pointer(--> Pointer[uint8]) {
        nativecast(Pointer[uint8], $!buf);
    }
    method write-pointer(--> Pointer) {
        Pointer[uint8].new: +$.buf-pointer + $!buf-len;
    }
    our method write(Pointer $in, uint32 $len --> int32)
            note "writing $len bytes";
    }
 }
```

Workplace Tool?

Development usage. NativeCall, etc for:

- Testing
- Prototyping
- Refactoring
- Migration?

Perl 6 Books are on the way

- Think Perl 6: How to Think Like a Computer Scientist, by Laurent Rosenfeld (published, print)
- Perl 6 Fundamentals , by Moritz Lenz (in work, can be bought right now, e-book)
- Learning Perl 6, by Brian D. Foy (in work)
- Migrating to Perl 6, by Andrew Shitov (in work)
- Web Application Development in Perl 6, by Gabor Szabo (draft, fundraising)