binding C libraries

A journey

why?

how?

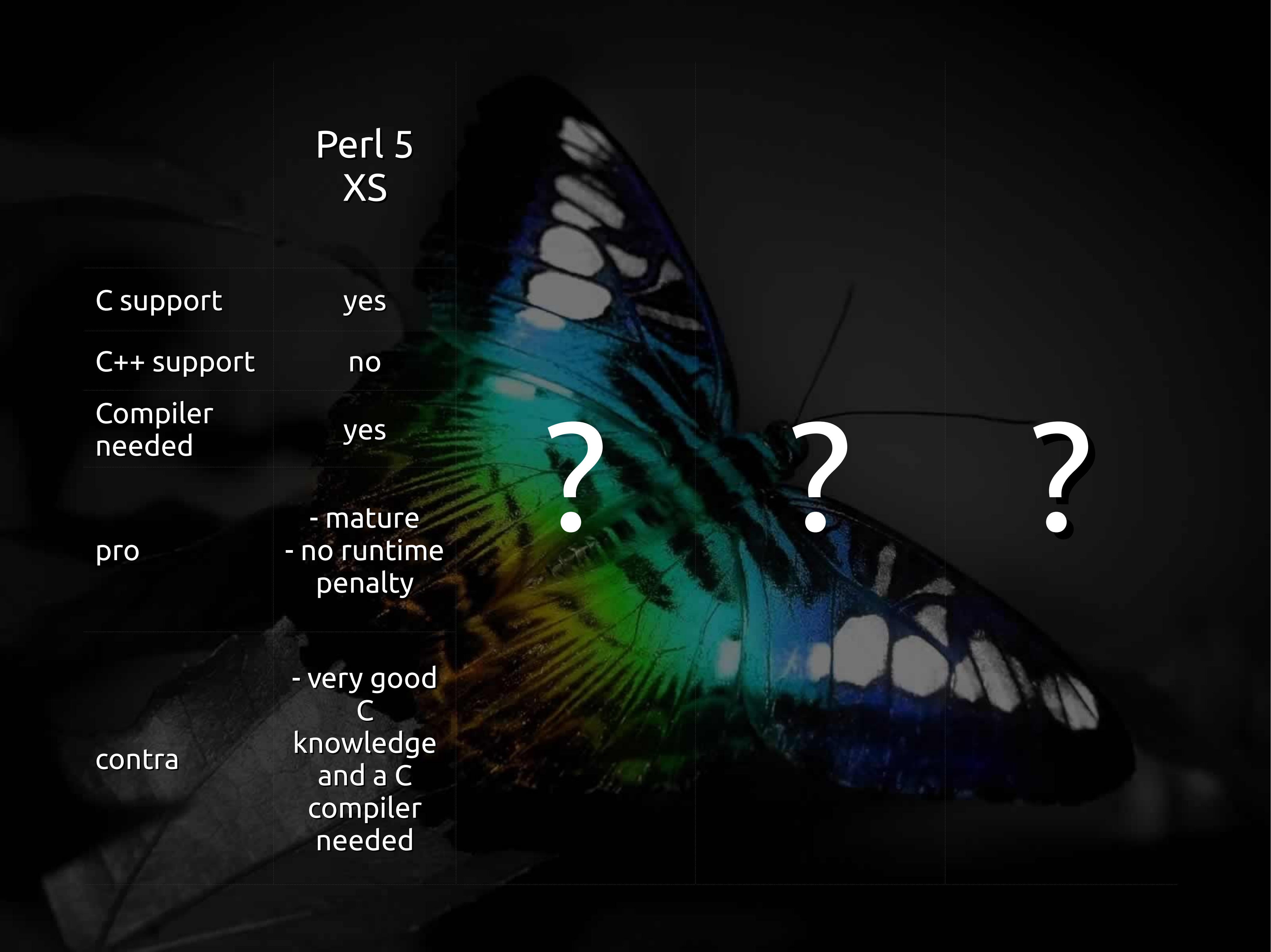
Perl 5 XS

```
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"
#include "ppport.h"
#include <SDL.h>

MODULE = SDL::Rect      PACKAGE = SDL::Rect    PREFIX = rect_

SDL_Rect *
rect_new (CLASS, x, y, w, h)
    char* CLASS
    Sint16 x, y
    Uint16 w, h
    CODE:
        RETVAL = (SDL_Rect *)safemalloc(sizeof(SDL_Rect));
        RETVAL->x = x;
        RETVAL->y = y;
        RETVAL->w = w;
        RETVAL->h = h;
    OUTPUT:
        RETVAL
```

```
Sint16
rect_x ( rect, ... )
    SDL_Rect *rect
    CODE:
        if (items > 1) rect->x = SvIV(ST(1));
        RETVAL = rect->x;
    OUTPUT:
        RETVAL

Sint16
rect_y ( rect, ... )
    SDL_Rect *rect
    CODE:
        if (items > 1) rect->y = SvIV(ST(1));
        RETVAL = rect->y;
    OUTPUT:
        RETVAL
```

```
SV *
createDocument( CLASS, version="1.0", encoding=NULL )
        char * version
        char * encoding
    ALIAS:
        XML::LibXML::Document::new = 1
    PREINIT:
        xmlDocPtr doc=NULL;
    CODE:
        PERL_UNUSED_VAR(ix);
        doc = xmlNewDoc((const xmlChar*)version);
        if (encoding && *encoding != 0) {
            doc->encoding = (xmlChar *)xmlStrdup((xmlChar *)encoding);
        }
        RETVAL = PmmNodeToSv(INT2PTR(xmlNodePtr,doc),NULL);
    OUTPUT:
        RETVAL
```

```
void
lmx_add( manager, bag )
   SDLx_LayerManager *manager
   SV* bag
   CODE:
      if( sv_isobject(bag) && (SvTYPE(SvRV(bag)) == SVt_PVMG) )
      {
         SDLx_Layer *layer    = (SDLx_Layer *)bag2obj(bag);
         layer->index         = av_len( manager->layers ) + 1;
         layer->manager       = manager;
         layer->touched       = 1;
         av_push( manager->layers, bag);
         SvREFCNT_inc(bag);
      }
```



	Perl 5 XS			
C support	yes			
C++ support	no			
Compiler needed	yes	?	?	?
pro	- mature - no runtime penalty			
contra	- very good C knowledge and a C compiler needed			

Perl 5 XSpp

```
%{
   #include <Box2D/Box2D.h>
%}

%module{Box2D};

%name{Box2D::b2Shape} class b2Shape
{
%{

void
b2Shape::ComputeAABB( aabb, xf )
      b2AABB* aabb
      b2Transform* xf
   CODE:
      THIS->ComputeAABB( aabb, *xf );

%}
};
```

int16	T_IV
uint16	T_IV
int32	T_IV
uint32	T_IV
float32	T_NV
b2Vec2 *	O_OBJECT
b2Mat22 *	O_OBJECT
b2World *	O_OBJECT
b2Body *	O_OBJECT
b2BodyDef *	O_OBJECT
b2Shape *	O_OBJECT
b2PolygonShape *	O_OBJECT
b2CircleShape *	O_OBJECT
b2Filter *	O_OBJECT
b2FixtureDef *	O_OBJECT
b2Fixture *	O_OBJECT
b2Transform *	O_OBJECT
b2Joint *	O_OBJECT
b2JointDef *	O_OBJECT
b2DistanceJoint *	O_OBJECT
b2DistanceJointDef *	O_OBJECT

	Perl 5 XS	Perl 5 XSpp		
C support	yes	no	?	?
C++ support	no	yes		
Compiler needed	yes	yes		
pro	- mature - no runtime penalty	- mature - no runtime penalty		
contra	- very good C knowledge and a C compiler needed	- special interface file syntax - compiler needed - no C support		

Perl 5 SWIG

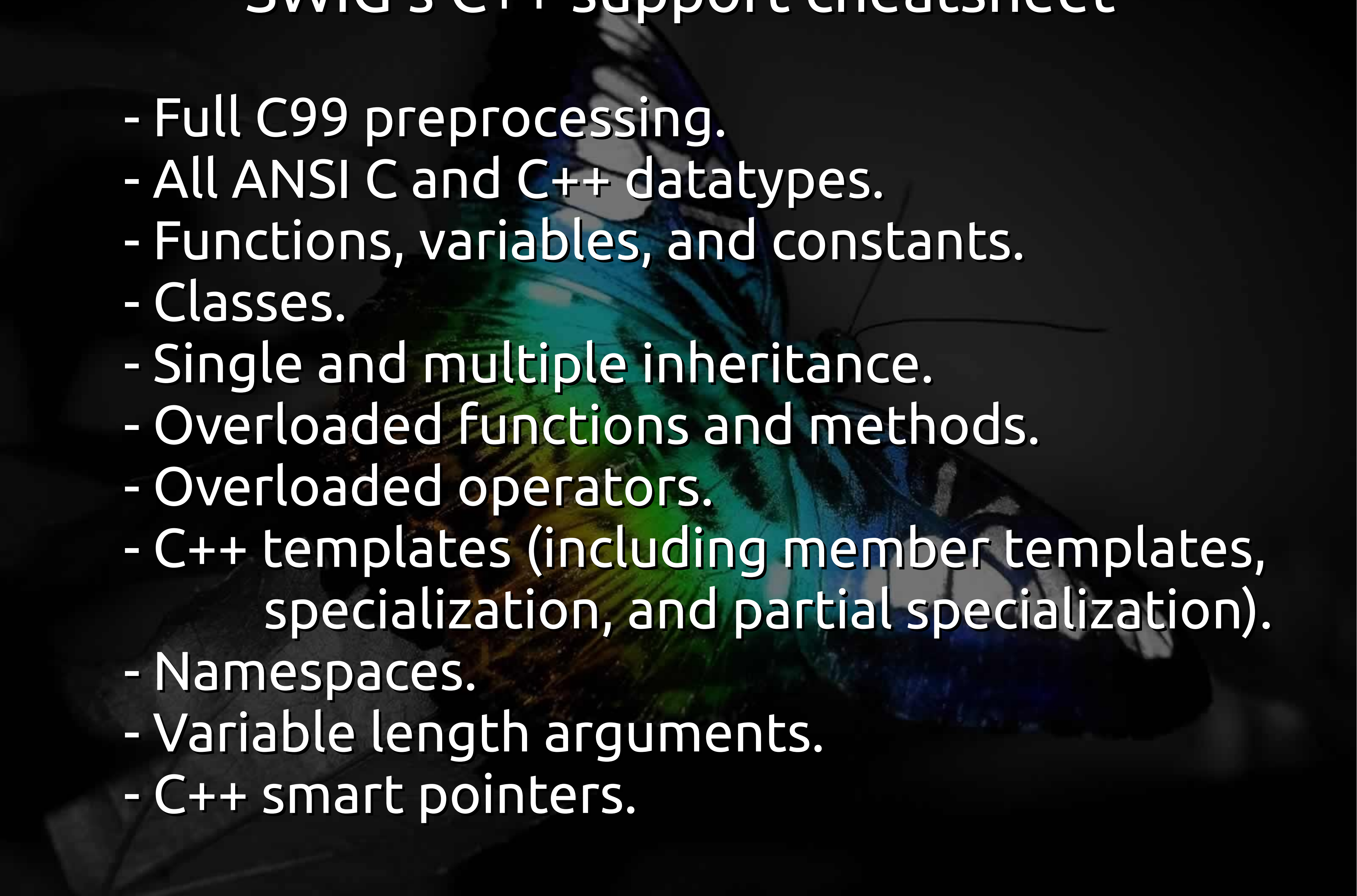
```c
/* File : example.c */

double  My_variable  = 3.0;

/* Compute factorial of n */
int  fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}


/* Compute n mod m */
int my_mod(int n, int m) {
    return(n % m);
}
```

```
/* File : example.i */
%module example
%{
/* Put headers and other declarations here */
extern double My_variable;
extern int    fact(int);
extern int    my_mod(int n, int m);
%}

extern double My_variable;
extern int    fact(int);
extern int    my_mod(int n, int m);
```

```
unix > swig -perl5 example.i
unix > gcc -c example.c example_wrap.c \
         -I/usr/local/lib/perl5/sun4-solaris/5.003/CORE
unix > ld -G example.o example_wrap.o -o example.so
unix > # ^--- This is for Solaris
unix > perl5.003
use example;
print example::fact(4), "\n";
print example::my_mod(23,7), "\n";
print $example::My_variable + 4.5, "\n";
<ctrl-d>
24
2
7.5
```

```
unix > swig -python example.i
unix > gcc -c -fpic example.c example_wrap.c \
         -I/usr/local/include/python2.0
unix > gcc -shared example.o example_wrap.o \
         -o _example.so
unix > python
Python 2.0 (#6, Feb 21 2001, 13:29:45)
[GCC egcs-2.91.66 19990314/Linux (egcs-1.1.2
release)] on linux2
Type "copyright", "credits" or "license" for more
information.
>>> import example
>>> example.fact(4)
24
>>> example.my_mod(23,7)
2
>>> example.cvar.My_variable + 4.5
7.5
```

```
unix > swig -perl5 -module example example.h
unix > gcc -c example.c example_wrap.c \
         -I/usr/local/lib/perl5/sun4-solaris/5.003/CORE
unix > ld -G example.o example_wrap.o -o example.so
unix > perl5.003
use example;
print example::fact(4), "\n";
print example::my_mod(23,7), "\n";
print $example::My_variable + 4.5, "\n";
<ctrl-d>
24
2
7.5
```



SWIG's C++ support cheatsheet

- Full C99 preprocessing.
- All ANSI C and C++ datatypes.
- Functions, variables, and constants.
- Classes.
- Single and multiple inheritance.
- Overloaded functions and methods.
- Overloaded operators.
- C++ templates (including member templates,
  specialization, and partial specialization).
- Namespaces.
- Variable length arguments.
- C++ smart pointers.

	Perl 5 XS	Perl 5 XSpp	Perl 5 SWIG	
C support	yes	no	yes	
C++ support	no	yes	yes	
Compiler needed	yes	yes	yes	
pro	- mature - no runtime penalty	- mature - no runtime penalty	- C and C++ support - interface file for other languages usable	?
contra	- very good C knowledge and a C compiler needed	- special interface file syntax - compiler needed - no C support	special interface file syntax or header files needed, compiler needed, edge cases problematic	

Perl 6 NativeCall

```c
/* File : example.c */

double  My_variable  = 3.0;

/* Compute factorial of n */
int  fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}


/* Compute n mod m */
int my_mod(int n, int m) {
    return(n % m);
}
```

```
use v6;
use NativeCall;

my $var = cglobal('example', 'My_variable', num64);

sub fact(int32) returns int32
                    is native('example') { * }

sub my-mod(int32, int32) returns int32
                    is symbol('my_mod')
                    is native('example') { * }

say fact 4;             # 24
say my-mod 23, 7; # 2
say $var + 4.5;        # 7.5
```

```
use v6;
use NativeCall;

my $var = cglobal('example', 'My_variable', num64);

say $var
```

```
use v6;
use NativeCall;

sub fact(int32)  returns int32  is native('example') { * }

say fact 4
```

```
use v6;
use NativeCall;

sub fact(int32)  returns int32  is native('example') { * }

say fact 4
```

```
sub my-mod(int32, int32) returns int32
                        is symbol('my_mod')
                        is native('example') { * }

say my-mod 23, 7
```

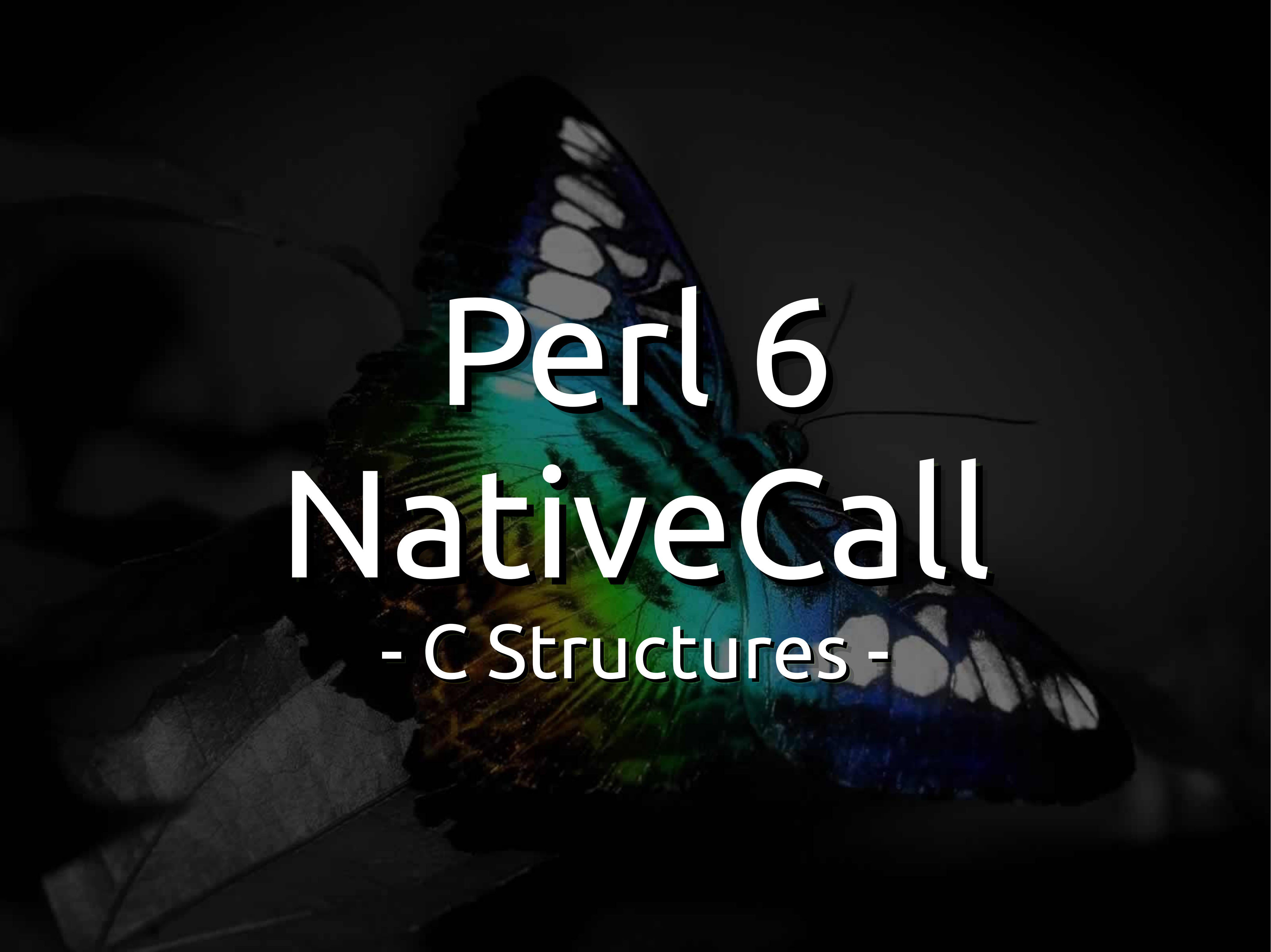
```
# Symbol              -> exposed as
SDL_BlitSurface  -> blit or blit-surface
SDL_FillRect         -> fill or fill-rect
xmlC14NDocDumpMemory -> ???
```

```
# Symbol              -> exposed as
SDL_BlitSurface  -> blit or blit-surface
SDL_FillRect         -> fill or fill-rect
xmlC14NDocDumpMemory -> ???
```

```
use v6;
use NativeCall;

sub fact(int32)  returns int32  is native('example') { * }

say fact 4
```

int8	char in C
int16	short in C
int32	int in C
int	32- or 64-bit, depends what long means locally
Int	always 64-bit, long long in C
num32	float in C
num64	double in C
num	same as num64
Str	C string
OpaquePointer	void *
CArray[Str]	char *foo[n]

```
use v6;
use NativeCall;

sub split(Str, int32 $limit = 42)
     returns CArray[Str]
     is native('splitter') { * }

say split('foobar')[3] # „b“
```



Perl 6 NativeCall

- C Structures -

```c
/**
 * xmlNs:
 *
 * An XML namespace.
 * Note that prefix == NULL is valid, it defines the default namespace
 * within the subtree (until overridden).
 *
 * xmlNsType is unified with xmlElementType.
 */

typedef struct _xmlNs xmlNs;
typedef xmlNs *xmlNsPtr;
struct _xmlNs {
    struct _xmlNs  *next;    /* next Ns link for this node  */
    xmlNsType      type;/* global or local */
    const xmlChar *href;     /* URL for the namespace */
    const xmlChar *prefix; /* prefix for the namespace */
    void           *_private;   /* application data */
    struct _xmlDoc *context;          /* normally an xmlDoc */
};
```

```
typedef struct _xmlNs xmlNs;
struct _xmlNs {
    struct _xmlNs          *next;     /* next Ns link for this node  */
    xmlNsType               type;     /* global or local */
    const xmlChar          *href;     /* URL for the namespace */
    const xmlChar        *prefix;     /* prefix for the namespace */
    void               *_private;     /* application data */
    struct _xmlDoc *context;          /* normally an xmlDoc */
};
```

```
typedef struct _xmlNs xmlNs;
struct _xmlNs {
   struct _xmlNs        *next;    /* next Ns link for this node  */
   xmlNsType             type;    /* global or local */
   const xmlChar        *href;    /* URL for the namespace */
   const xmlChar      *prefix;    /* prefix for the namespace */
   void             *_private;    /* application data */
   struct _xmlDoc *context;       /* normally an xmlDoc */
};

my class xmlNs is repr('CStruct') {
   has xmlNs                  $.next;      # next Ns link for this node
   has int8                   $.type;      # global or local
   has Str                     $.uri;      # URL for the namespace
   has Str                    $.name;      # prefix for the namespace
   has OpaquePointer $._private;           # application data
   has xmlDoc              $.context;      # normally an xmlDoc
}
```

```
my class xmlNs is repr('CStruct') {
   has xmlNs                      $.next;      # next Ns link for this node
   has int8                       $.type;      # global or local
   has Str                         $.uri;      # URL for the namespace
   has Str                        $.name;      # prefix for the namespace
   has OpaquePointer $._private;      # application data
   has xmlDoc                  $.context;      # normally an xmlDoc
}

#`( Search a Ns aliasing a given URI. Recurse on the parents
     until it finds the defined namespace or return NULL
     otherwise. )
sub xmlSearchNsByHref(xmlDoc, xmlNode, Str)
        returns xmlNs  is native('libxml2') is export { * }
```

```
my class xmlNs is repr('CStruct') {
   has xmlNs                     $.next;      # next Ns link for this node
   has int8                      $.type;      # global or local
   has Str                        $.uri;      # URL for the namespace
   has Str                       $.name;      # prefix for the namespace
   has OpaquePointer $._private;              # application data
   has xmlDoc                 $.context;      # normally an xmlDoc
}

#`( Search a Ns aliasing a given URI. Recurse on the parents
    until it finds the defined namespace or return NULL
    otherwise. )
sub xmlSearchNsByHref(xmlDoc, xmlNode, Str)
         returns xmlNs  is native('libxml2') is export { * }

my $ns = xmlSearchNsByHref($node.doc, $node, 'foo');
say $ns
```

```
my class xmlNs is repr('CStruct') {
   has xmlNs                    $.next;     # next Ns link for this node
   has int8                     $.type;     # global or local
   has Str                       $.uri;     # URL for the namespace
   has Str                      $.name;     # prefix for the namespace
   has OpaquePointer $._private;            # application data
   has xmlDoc                $.context;     # normally an xmlDoc
}

#`( Search a Ns aliasing a given URI. Recurse on the parents
    until it finds the defined namespace or return NULL
    otherwise. )
sub xmlSearchNsByHref(xmlDoc, xmlNode, Str)
        returns xmlNs  is native('libxml2') is export { * }

my $ns = xmlSearchNsByHref($node.doc, $node, 'foo');
say $ns # „(xmlNs)“
```

```
my class xmlNs is repr('CStruct') {
   has xmlNs                   $.next;      # next Ns link for this node
   has int8                    $.type;      # global or local
   has Str                      $.uri;      # URL for the namespace
   has Str                     $.name;      # prefix for the namespace
   has OpaquePointer       $._private;      # application data
   has xmlDoc               $.context;      # normally an xmlDoc
}

#`( Search a Ns aliasing a given URI. Recurse on the parents
    until it finds the defined namespace or return NULL
    otherwise. )
sub xmlSearchNsByHref(xmlDoc, xmlNode, Str)
        returns xmlNs  is native('libxml2') is export { * }

my $ns = xmlSearchNsByHref($node.doc, $node, 'bar');
say $ns.name # „baz“
```

```
my class xmlNs is repr('CStruct') {
   has xmlNs                    $.next;     # next Ns link for this node
   has int8                     $.type;     # global or local
   has Str                       $.uri;     # URL for the namespace
   has Str                      $.name;     # prefix for the namespace
   has OpaquePointer        $._private;     # application data
   has xmlDoc                $.context;     # normally an xmlDoc
}

# version string of the document the namespace belongs to
say $ns.context.version # „1.0“
```

```
/*
 * A node-set (an unordered collection of nodes without duplicates).
 */
typedef struct _xmlNodeSet xmlNodeSet;
typedef xmlNodeSet *xmlNodeSetPtr;
struct _xmlNodeSet {
    int nodeNr;                 /* number of nodes in the set */
    int nodeMax;                /* size of the array as allocated */
    xmlNodePtr *nodeTab;        /* array of nodes in no particular order */
};
```

```
/*
 * A node-set (an unordered collection of nodes without duplicates).
 */
typedef struct _xmlNodeSet xmlNodeSet;
typedef xmlNodeSet *xmlNodeSetPtr;
struct _xmlNodeSet {
   int nodeNr;                    /* number of nodes in the set */
   int nodeMax;                   /* size of the array as allocated */
   xmlNodePtr *nodeTab;           /* array of nodes in no particular order */
};

my class xmlNodeSet is repr('CStruct') {
   has int32                 $.nodeNr;   # number of nodes in the set
   has int32                $.nodeMax;   # size of the array as allocated
   has CArray[xmlNode] $.nodeTab;   # array of nodes in no particular ...
}
```

```
typedef struct _xmlNodeSet xmlNodeSet;
typedef xmlNodeSet *xmlNodeSetPtr;
struct _xmlNodeSet {
   int nodeNr;                    /* number of nodes in the set */
   int nodeMax;                   /* size of the array as allocated */
   xmlNodePtr *nodeTab;       /* array of nodes in no particular order */
};

my class xmlNodeSet is repr('CStruct') {
   has int32                  $.nodeNr;   # number of nodes in the set
   has int32                 $.nodeMax;  # size of the array as allocated
   has CArray[xmlNode] $.nodeTab;  # array of nodes in no particular ...
}

for ^$set.nodeNr -> $idx {
   say $set.noteTab[$idx].value
}
```

```
typedef struct _xmlNodeSet xmlNodeSet;
typedef xmlNodeSet *xmlNodeSetPtr;
struct _xmlNodeSet {
    int nodeNr;                    /* number of nodes in the set */
    int nodeMax;                   /* size of the array as allocated */
    xmlNodePtr *nodeTab;           /* array of nodes in no particular order */
};

my class xmlNodeSet is repr('CStruct') {
    has int32                  $.nodeNr;   # number of nodes in the set
    has int32                 $.nodeMax;   # size of the array as allocated
    has CArray[xmlNode] $.nodeTab;   # array of nodes in no particular ...
}

for ^$set.nodeNr -> $idx {
    say $set.noteTab[$idx].value
}

$set.noteTab[^$set.nodeNr]».value».say
```

Perl 6 NativeCall

- Enumerations -

```
/*
 * xmlC14NMode:
 *
 * Predefined values for C14N modes
 *
 */
typedef enum {
   XML_C14N_1_0                = 0,  /* Origianal C14N 1.0 spec */
   XML_C14N_EXCLUSIVE_1_0  = 1,  /* Exclusive C14N 1.0 spec */
   XML_C14N_1_1                = 2   /* C14N 1.1 spec */
} xmlC14NMode;
```

```
/*
 * xmlC14NMode:
 *
 * Predefined values for C14N modes
 *
 */
typedef enum {
   XML_C14N_1_0                      = 0,  /* Origianal C14N 1.0 spec */
   XML_C14N_EXCLUSIVE_1_0  = 1,  /* Exclusive C14N 1.0 spec */
   XML_C14N_1_1                      = 2   /* C14N 1.1 spec */
} xmlC14NMode;

enum xmlC14NMode (
   XML_C14N_1_0                    => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1                    => 2, # C14N 1.1 spec
);
```

```
enum xmlC14NMode (
   XML_C14N_1_0                => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1                => 2, # C14N 1.1 spec
);
```

```
enum xmlC14NMode (
   XML_C14N_1_0           => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1           => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“
```

```
enum xmlC14NMode (
   XML_C14N_1_0           => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1           => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“
```

```
enum xmlC14NMode (
   XML_C14N_1_0             => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1             => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“

say XML_C14N_EXCLUSIVE_1_0.WHAT
   # „(xmlC14NMode)“
```

```
enum xmlC14NMode (
   XML_C14N_1_0           => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1           => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“

say XML_C14N_EXCLUSIVE_1_0.WHAT
   # „(xmlC14NMode)“

say XML_C14N_EXCLUSIVE_1_0.perl
   # “xmlC14NMode::XML_C14N_EXCLUSIVE_1_0“
```

```
enum xmlC14NMode (
   XML_C14N_1_0            => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1            => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“

say XML_C14N_EXCLUSIVE_1_0.WHAT
   # „(xmlC14NMode)“

say XML_C14N_EXCLUSIVE_1_0.perl
   # “xmlC14NMode::XML_C14N_EXCLUSIVE_1_0“

sub foo(xmlC14NMode $mode) { ... }
```

```
enum xmlC14NMode (
   XML_C14N_1_0            => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1            => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“

say XML_C14N_EXCLUSIVE_1_0.WHAT
   # „(xmlC14NMode)“

say XML_C14N_EXCLUSIVE_1_0.perl
   # “xmlC14NMode::XML_C14N_EXCLUSIVE_1_0“

sub foo(xmlC14NMode $mode) { … }
sub foo(xmlC14NMode $mode = XML_C14N_1_1) { … }
```

```
enum xmlC14NMode (
   XML_C14N_1_0             => 0, # Origianal C14N 1.0 spec
   XML_C14N_EXCLUSIVE_1_0 => 1, # Exclusive C14N 1.0 spec
   XML_C14N_1_1             => 2, # C14N 1.1 spec
);

say +XML_C14N_EXCLUSIVE_1_0 # „1“

say xmlC14NMode(1) # „XML_C14N_EXCLUSIVE_1_0“

say XML_C14N_EXCLUSIVE_1_0.WHAT
   # „(xmlC14NMode)“

say XML_C14N_EXCLUSIVE_1_0.perl
   # “xmlC14NMode::XML_C14N_EXCLUSIVE_1_0“

sub foo(xmlC14NMode $mode) { ... }
sub foo(xmlC14NMode $mode = XML_C14N_1_1) { ... }

foo(XML_C14N_EXCLUSIVE_1_0) # does stuff
```

Perl 6 NativeCall

- Casting -

```
struct Foo {
   void       *theObject;
   objectType      type; /* enum Bar, Baz, … */
};

my class Foo is repr('CStruct') {
   has OpaquePointer $.obj;
   has int8               $.type;
}

say $foo.obj;                     # „OpaquePointer<0x529312>“
say objectType($foo.obj);   # „Bar“

if $foo.obj -> $o {
   my $bar = nativecast(objectType($o), $o);
   # do something …
}
```

Perl 6 NativeCall

- Callbacks -

```c
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
          const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);
```

```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
          const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
```

```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
            const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
"messwithit(/foo/bar[1])"
```

```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
            const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
"messwithit(/foo/bar[1])"

sub xmlXPathRegisterFunc(xmlXPathContext, ?) is native('libxml2') { * }
```

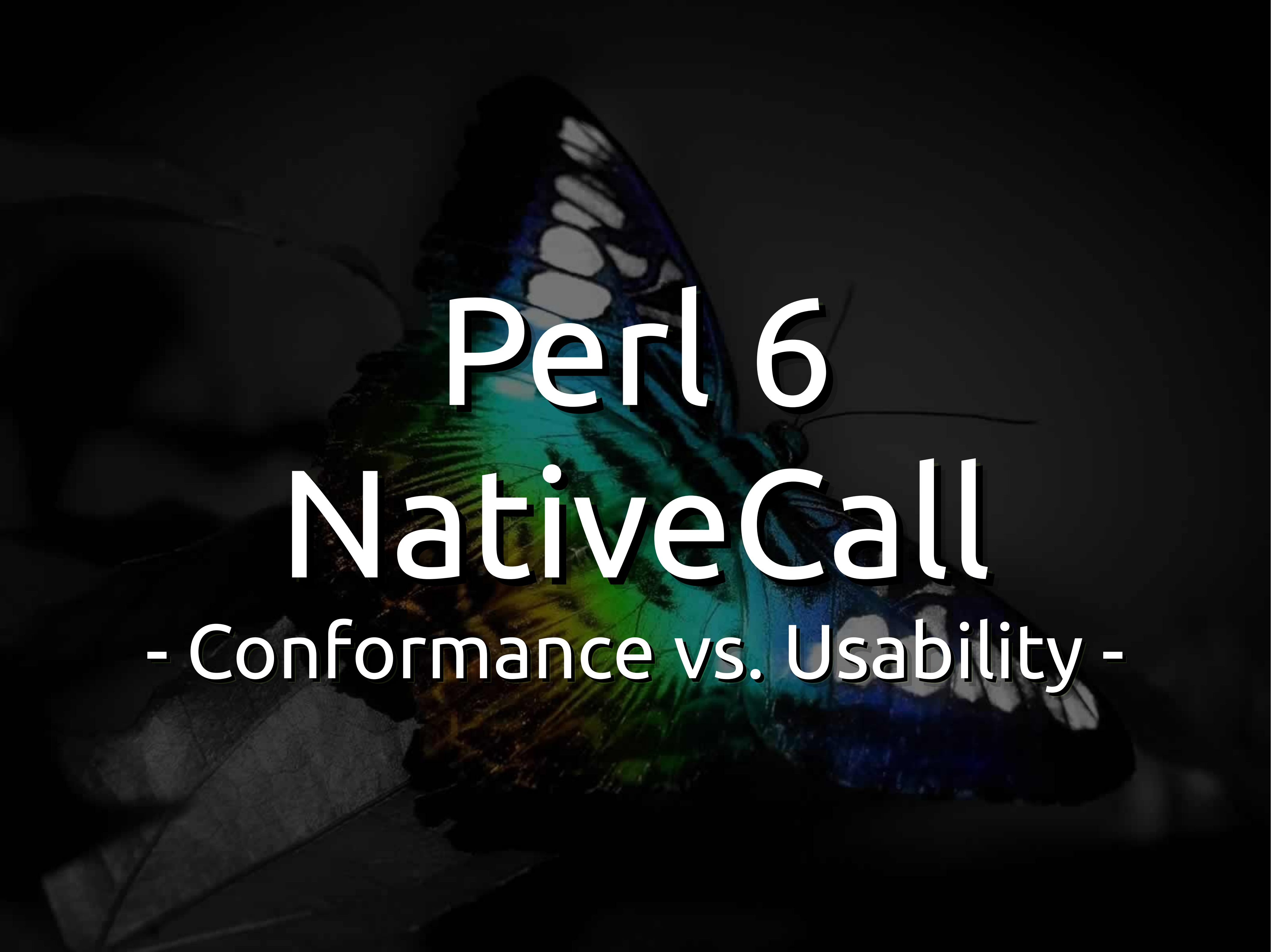
```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
          const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
"messwithit(/foo/bar[1])"

sub xmlXPathRegisterFunc(xmlXPathContext,
 &custom-xpath-func (xmlXPathContext, int32) ) is native('libxml2') { * }
```

```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
           const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
"messwithit(/foo/bar[1])"

sub xmlXPathRegisterFunc(xmlXPathContext,
 &custom-xpath-func (xmlXPathContext, int32) ) is native('libxml2') { * }

sub messwithit(xmlXPathContext $ctxt, int32 $nargs) { #`( stuff ) }
```

```
/* Register a new function. If @f is NULL it unregisters the function */
int  xmlXPathRegisterFunc(xmlXPathContextPtr ctxt,
          const xmlChar * name, xmlXPathFunction f);

/* An XPath function. The arguments (if any) are popped out from the
 * context stack and the result is pushed on the stack. */
void xmlXPathFunction(xmlXPathParserContextPtr ctxt, int nargs);

"contains(/foo/bar[1], 'test 1')"
"messwithit(/foo/bar[1])"

sub xmlXPathRegisterFunc(xmlXPathContext,
 &custom-xpath-func (xmlXPathContext, int32) ) is native('libxml2') { * }

sub messwithit(xmlXPathContext $ctxt, int32 $nargs) { #`( stuff ) }

xmlXPathRegisterFunc($ctxt, &messwithit);
```

```
perl6 -MXML::LibXML -e 'say parse-xml "<fooo><bar/></foo>"'
```

```
perl6 -MXML::LibXML -e 'say parse-xml "<fooo><bar/></foo>"'
===SORRY!=== Error while parsing XML document
XML::LibXML::Parser error: Extra content at the end of the document
<fooo><bar/><⏏/foo>
  in method gist at src/gen/m-CORE.setting:14570
  in sub say at src/gen/m-CORE.setting:17327
  in block <unit> at -e:1
```



Perl 6 NativeCall

- Conformance vs. Usability -

```
$xml_doc->documentElement
        ->firstChild
        ->toStringC14N(1)
```

```
$xml-doc.documentElement\
        .firstChild\
        .toStringC14N(:comments)
```

```
$xml-doc.document-element\
        .first-child\
        .c14n(:comments)
```

```
$xml-doc.document-element[0]
         .c14n(:comments)
```

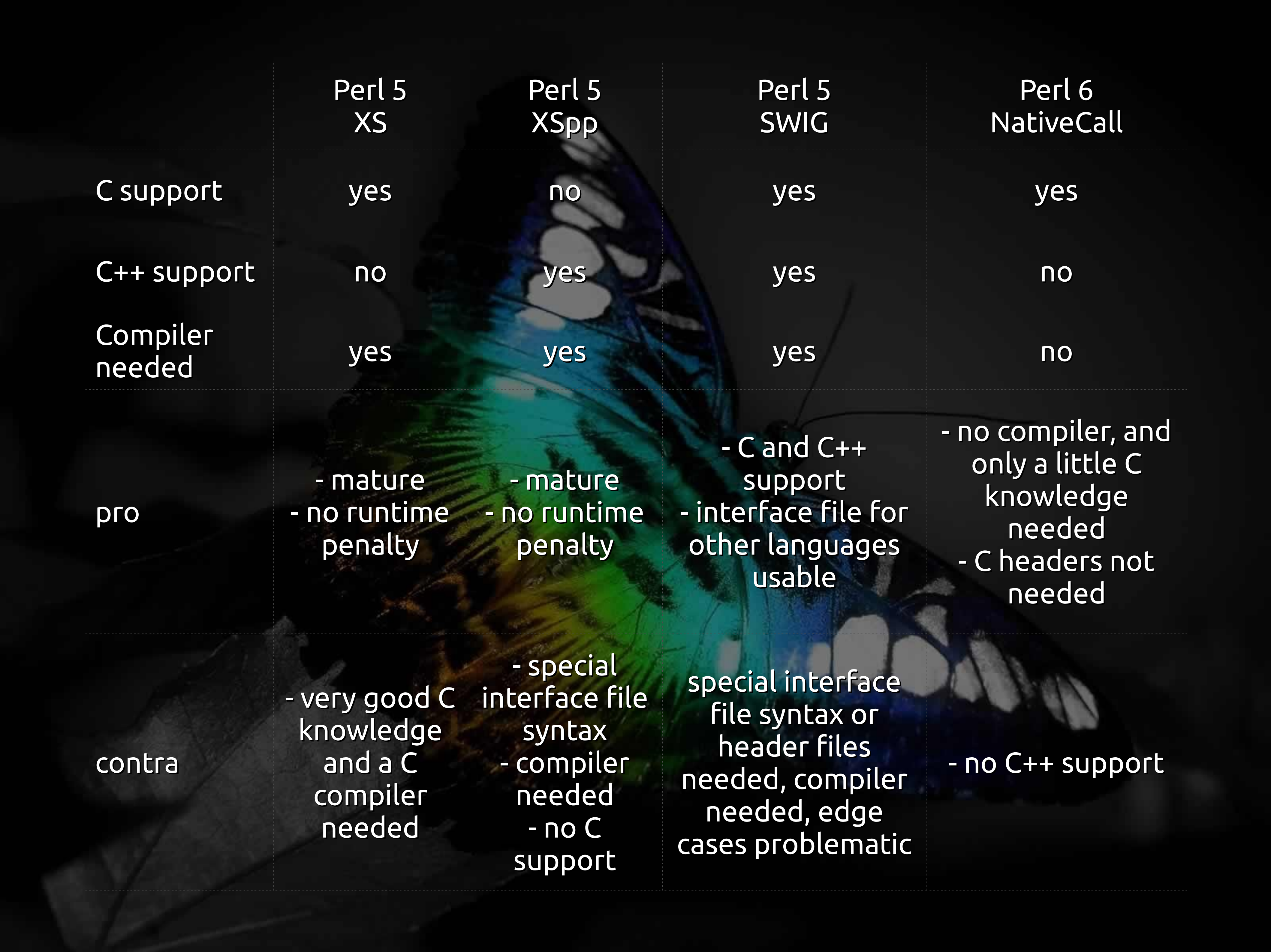
A Rant

Function: xmlXPathNewString

```
xmlXPathObjectPtr   xmlXPathNewString  (const xmlChar * val)
```

Create a new xmlXPathObjectPtr of type string and of value @val

val:	the xmlChar * value
Returns:	the newly created object.



	Perl 5 XS	Perl 5 XSpp	Perl 5 SWIG	Perl 6 NativeCall
C support	yes	no	yes	yes
C++ support	no	yes	yes	no
Compiler needed	yes	yes	yes	no
pro	- mature - no runtime penalty	- mature - no runtime penalty	- C and C++ support - interface file for other languages usable	- no compiler, and only a little C knowledge needed - C headers not needed
contra	- very good C knowledge and a C compiler needed	- special interface file syntax - compiler needed - no C support	special interface file syntax or header files needed, compiler needed, edge cases problematic	- no C++ support

FROGGS
freenode/#perl6
FROGGS@cpan.org
github.com/FROGGS

- simple example
- the is native trait
- the empty body
- the is symbol trait
- the signature
- type mapping
- callbacks
- the returns trait

- cglobal
- nativecast
- comparision
- works on Parrot, JVM and MoarVM
- enums
- is encoded('utf8')