Protocol

- Encoding https://developers.google.com/protocol-buffers/docs/encoding
- Protocol Buffer Polymorphism http://www.indelible.org/ink/protobuf-polymorphism/

tools

http://yura415.github.io/js-protobuf-encode-decode/

Libraries

sean-lin based

https://github.com/sean-lin/protoc-gen-lua

- Latest commit 12 Jun 2014

https://github.com/djungelorm/protobuf-lua

- Latest commit **26 Jan 2016**
- Latest fixes https://github.com/skwerlman/protobuf-lua/tree/patch-1
- Lua rocks: https://luarocks.org/modules/djungelorm/protobuf
- Prerequisities
  - `sudo apt-get install python-protobuf`
- protoc plugin used: `/usr/local/bin/protoc-gen-lua` which is symlinked to `/usr/local/lib/luarocks/rocks/protobuf/1.1.1-0/protoc-plugin/protoc-gen-lua`

https://github.com/urbanairship/protobuf-lua

- Latest commit 13 Nov 2015
- fork of above https://github.com/djungelorm/protobuf-lua
- Lua 5.2 support
- .proto compilation required. Contains protoc plugin for lua

others

https://github.com/indygreg/lua-protobuf

- Last commit 27 Mar 2011

https://github.com/haberman/upb

- Last commit 8 Jul 2015
- Right now the Lua bindings support:
  - Building schema objects manually (eg. you can essentially write .proto files natively in Lua).
  - creating message objects.
  - parsing Protocol Buffers into message objects.

https://github.com/starwing/lua-protobuf

- Last commit on Aug 24, 2016

```
This project offers a simple C library for basic protobuf wire format
encode/decode, it splits to several modules:
  - pb.decoder: a wire format decode module.
  - pb.buffer:  a buffer implement that use to encode basic types into
                protobuf's wire format. It also can be used to support
        streaming decode protobuf data.
  - pb.conv:    a module to convert between integers in protobuf.
  - pb.io:      a module to support binary mode read/write to stdin/stdout.
```

Neopalium

https://github.com/Neopallium/lua-pb

- Latest commit 18 Feb 2016
- **Own LUA parser** for .proto files (no need to compile)

Library API

```lua
-- ------------------------------------------
-- @return string with message dump
local function dumpRawProtoc (binMsg)
  local p = io.popen ("protoc --decode_raw", "w")
  local stdout
  if p then
    p:write(binMsg)
    local stdout = p:read("*a")
    p:close()
  end
  return stdout
end
```

Neopalium

```lua
local pb = require 'pb'
```

```
-- loading 'pb' module will replace 'require' function
-- now .proto files are automatically loaded by require.
local proto = require 'protos.messages' -- load 'protos/messages.proto'

-- Two methods of message creation
-- 1) direct assign
local protoMsg = proto.Message()
protoMsg.deviceId = 0x1234   -- simple type value
protoMsg.deviceType = 'MODEM' -- enum value
protoMsg.repeatedSubMessage = { { id = 'TEMPERATURE',
                                  value = 34
                                },
                                { id = 'WIND',
                                  value = 2
                                }
                              }
protoMsg.subMessage.serialNumber = '123456-1234'
protoMsg.subMessage.version      = '2'

-- 2) Initialize from structure
local luaData = {
  deviceId = 0x1234,
  deviceType = 'MODEM',
  repeatedSubMessage = { { id = 'TEMPERATURE',
                           value = 34
                         },
                         { id = 'WIND',
                           value = 2
                         }
                       },
   subMessage = { serialNumber = '123456-1234',
                  version      = '2'
                }
}

local protoMsg = proto.Message(luaData) -- initialize from luaData

-- 3) Still possible to directly modify, add values into message
protoMsg.deviceId = 0xdeadbeef


local binProtoMsg = protoMsg:Serialize() -- binProtoMsg is string
print (binProtoMsh:len())
local dumpProtoMsg = protoMsg:SerializePartial('text')


local receivedProtoMsg = proto.Message()
if receivedProtoMsh == nil then
  error ("Malformed message")
end
```

```
receivedProtoMsg:Parse(binProtoMsg)
print (receivedProtoMsg:SerializePartial('text'))
```

Raw dump of protobuf message (.proto files not needed)

```
local function dump_fields(unknown)
    local o = ""
    for i, v in ipairs(unknown) do
      o = o..string.format("#%02d Tag=[%2d] Wire=[%2d] %q\n", i, v.tag,
v.wire, tostring(v.value))
      if type (v.value) == 'table' then
        o = o..dump_fields(v.value)
      end
    end
    return o
end

-- ---------------------------------------------
-- @return string with message dump
function PD:pbDumpRawPB(binMsg)
  local msg, off = pb.decode_raw(binMsg)
  return dump_fields(msg.unknown_fields)
end
```

Issue with 64bit number

18446744073709551615 is exactly 2^64-1

in LuaJIT 64bit numbers are stored as double, so some precision is lost:

```
print (0xFFFFFFFFFFFFFFFF == 0xFFFFFFFFFFFFFF00)
true
```

If library detects LuaJIT with FFI support, it is using FFI cdata to store 64 bit numbers `int64_t` or `uint64_t` depends on `signed` argument passed to `make_int64_func`. If no LuaJIT is used, big values are represented as 8 bytes strings.

```
local pb = require"pb"
local make_int64 = pb.get_make_int64_func()
-- make_int64(b8,b7,b6,b5,b4,b3,b2,b1, signed)
fakeNodeId = make_int64(0,0,0,0,  0,0,0,1 )
-- or initialize from string
local v = \255\255\255\255\255\255\255\255"
fakeNodeId = make_int65(v:byte(1,8))
print (fakeNodeId)
-- prints: '1ULL'
fakeNodeIs = fakeNodeId * 22
print (fakeNodeId)
-- prints: '22ULL'
```

But library encoder doesn't work:

```
luaData = {
   ["deviceId"] = 1.844674407371e+19;
};

binProtoMsg=
  0000 08 00                                              ..

binProtoMsg decoded=deviceId: 0
```

djungelorm

No documentation. Poor examples. Need to dig in sources to familiarize.

First, proto files needs to be compile with `protoc` compile with lua output plugin. For `messages.proto` corresponding lua file `messages_pb.lua` will be created.

API:

- RCFC (Repeated Composite Field Container) Repeated fields cannot be modified directly by assignment. Use:
  - v:add() return newly vreated value object
  - v:remove(key)
- RSFC (Repeted Single Field Container)
  - v:append(value)
  - v:remove(key)

- Message methods:
  - _member:ListFields()
  - _member:HasField(field_name)
  - _member:ClearField(field_name)
  - For extendable messages only:
    - _member:ClearExtension (extension_handle)
    - _member:HasExtenstion (extension_handle)
  - _member:Clear()
  - :__tostring()
  - _member:SetListener(listener)
  - _member:ByteSize()
  - _member:SerializeToString()
  - _member:SerializePartialToString()
  - _member:MergeFromString(serialized)
  - _member:IsInitialized(errors) - errors - luaTable (numeric indexes)
  - _member:MergeFrom(msg)

Other internal methods:

- _member.RegisterExtension (extension_handle)
- _member.FromString(s)
- _member:SerializeToIOString(iostring)

- _member:_InternalSerialize(write_bytes)
- _member:SerializePartialToIOString(iostring)
- _member:ParseFromString(serialized)
- _member:FindInitalizationErrors()

```
p = require 'messages_pb'
local message = p.Message()
message.deviceId = 0xdeadbeef -- assign int value
message.code = p.Code.OK -- assign enum value (defined in Code enum)
message.repeatedSubMessage:add()
message.repeatedSubMessage[1].ts = 12345 -- assign int value
message.repeatedSubMessage:add()
message.repeatedSubMessage[2].ts = 6789 -- assign int value

print (message:SerializeToString())
```