What's in a Protobuf message?

Ronan McHugh, October 2020

What is Protobuf?

Protobuf is a binary encoding format

Encoding

Why do we encode?

Sharing memory

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)
- Sharing data within a program involves sharing access to these data structures (typically using pointers)

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)
- Sharing data within a program involves sharing access to these data structures (typically using pointers)
- p1 = Person.new(name: "Ronan", email: "romchugh@zendesk.com")

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)
- Sharing data within a program involves sharing access to these data structures (typically using pointers)
- p1 = Person.new(name: "Ronan", email: "romchugh@zendesk.com")
- BillingService.remind!(p1)

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)
- Sharing data within a program involves sharing access to these data structures (typically using pointers)
- p1 = Person.new(name: "Ronan", email: "romchugh@zendesk.com")
- BillingService.remind!(p1)
- We can only do this when we share memory

Why do we encode?

Sharing memory

- Internally: computer processes use data structures contained in memory (e.g. hash, list, string)
- Sharing data within a program involves sharing access to these data structures (typically using pointers)
- p1 = Person.new(name: "Ronan", email: "romchugh@zendesk.com")
- BillingService.remind!(p1)
- We can only do this when we share memory
- But what happens when we don't share memory?

Why do we encode?

Between processes

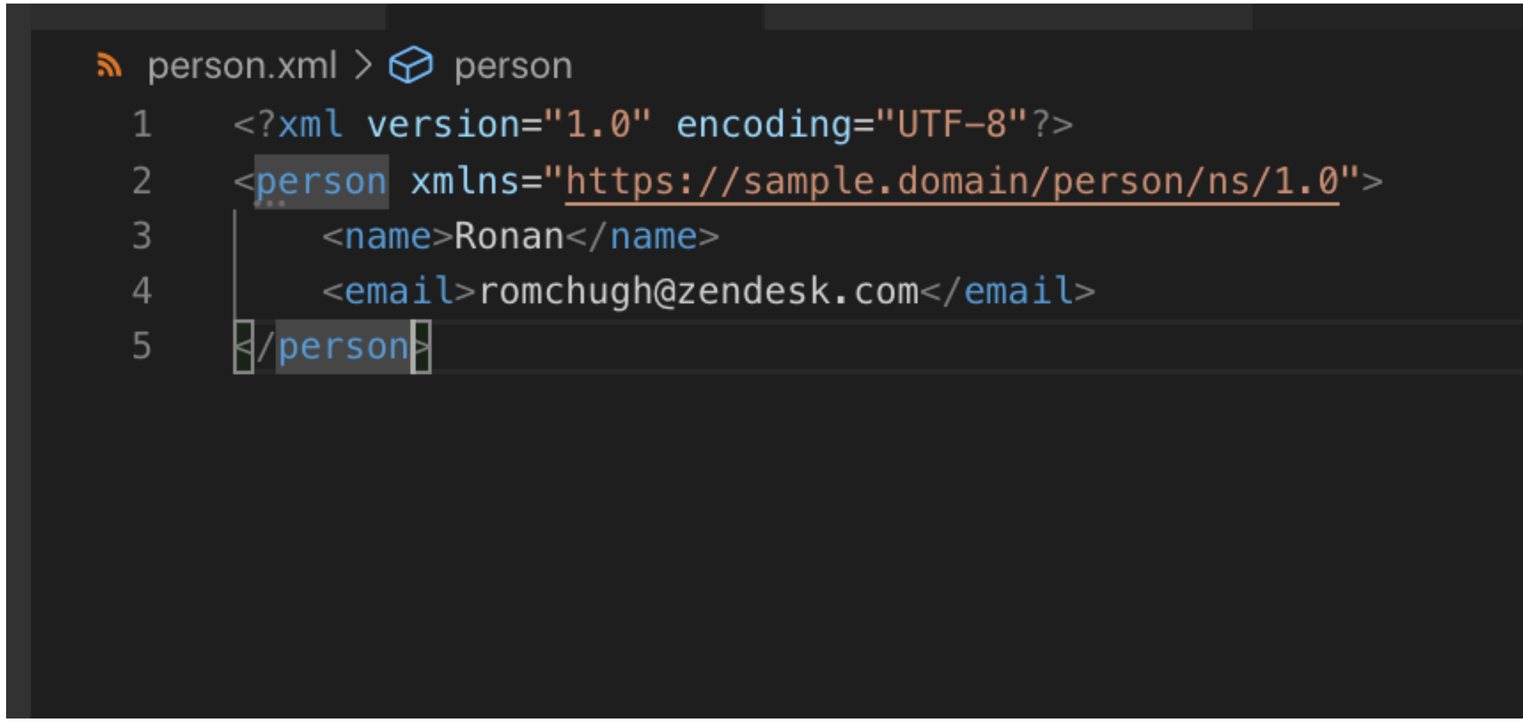
- If we don't have memory access we need to communicate via disk or via network.
- In either case, data must be encoded as a **stream of bytes**
- They can be sent as network packets, or written to disk as a file
- person = { name: "Ronan", email: "romchugh@zendesk.com" }
- File.write(person, JSON.dump(person))

Encoding

Textual encoding formats

XML

You may mock



164 Bytes

JSON

The old hotness

133 Bytes

Advantages & Disadvantages

- Self-describing -> Good for external endpoints
- Verbose -> Costly for large scale data transmission / storage

Encoding

Binary encoding formats

Binary Encoding

- Binary encoding allows data to be encoded in less bytes
- But encoding / decoding typically requires specialised libraries and often schemas

Protobuf

29 Bytes!

But also, wtf?

Let's break it down

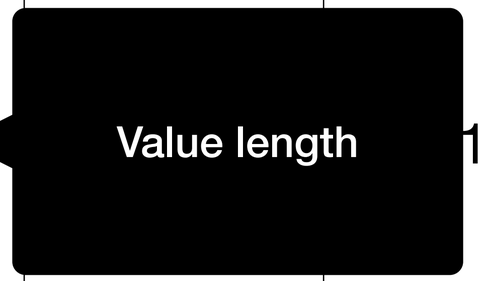
Bits	"00001010"	"00000101"	"01010010"	"01101111"	"01101110"	"01100001"	"01101110"
Decimal	10	5	82	111	110	97	110
Char			R	o	n	a	n

Bits	"00001010"	[illegible]	[illegible]10"	"01101111"	"01101110"	"01100001"	"01101110"
Decimal	10	5	82	111	110	97	110
Char			R	o	n	a	n

I am a field tag

Bits	"00001010"	"00000101"	"01010010"	"01101111"	"01101110"	"01100001"	"01101110"
Decimal	10	5	82	111	110	97	110
Char			R	o	n	a	n

Bits	"00001010"	"00000101"	"01010010"	"01101111"	"01101110"	"01100001"	"01101110"
Decimal	10	5		11	110	97	110
Char			R	o	n	a	n



What's in a field tag?

Bits	"0"	"0001"	"010"
Decimal	0	1	2
Role	MSB	Field Number	Wire Type
Meaning		1st Field	Length delimited

Wire types

Type	Meaning	Used for
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64 bit	fixed64, sfixed64, double
2	Length delimited	string, bytes, embedded messages, packed repeated fields
5	32 bit	fixed32, sfixed32, float

Varints

What's wrong with int64?

- Fixed length numbers have size constraints
- Always choosing large sizes would usually waste space
- What if the format can adapt to the size of the integer?

Varints

Bits	Decimal	Size
0000 0001	1	1 byte
1010 1100 0000 0010	300	2 bytes

Varints

- 1010 1100 0000 0010
- 010 1100 000 0010
- 000 0010 010 1100
- 100101100
- 256 + 32 + 8 + 4 = 300