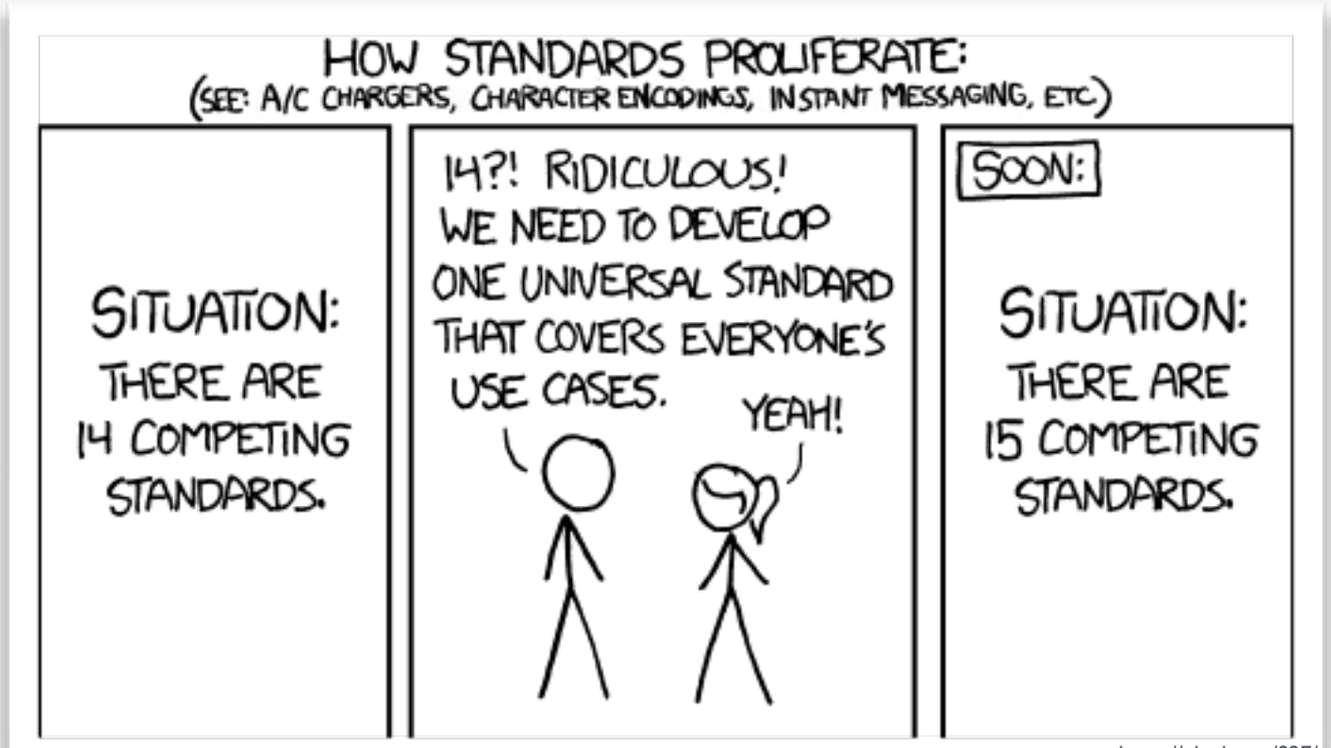
One BSON to Rule Them

David Golden
Staff Engineer, MongoDB

NY Perl Meetup • Dec 2016

This talk in a nutshell...



https://xkcd.com/927/

First... let's talk types

Know JSON?

```
{
   "name":"Larry Wall",
   "answer":42,
   "boolean":true,
   "nothing":null,
   "mixed_type_stuff":[
      "a",23,{"key":"value "}
   ]
}
```

MongoDB transmits data as 'BSON'

BSON = 'Binary JSON'

Like JSON,

except when not

JSON	BSON
schema-less	schema-less
key-value pairs	key-value pairs
text	binary
6 types	21 types
unordered	ordered

JSON/BSON types...

JSON	BSON
document	document
array	array
null	null
string	string
boolean	boolean
number	**int32**
	int64
	float64 (double)
	decimal128
	binary (blob)
	datetime
	regular expression
	javascript code
	javascript code with data
	"object id"
	"minimum key"
	"maximum key"
	"timestamp"
	(3 deprecated types)

Serializing Perl is hard

Start with JSON

JSON	Perl
document	hash reference
array	array reference
null	undef
string	**string**
number	**number**
boolean	**???**

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	

```
use JSON;
my $n = 23;

say encode_json({n => $n});
# {"n":23}
```

```
use JSON;
my $n = 23;

say encode_json({n => $n});
# {"n":23}
```

```
$n =~ /^\d+$/;
```

```
use JSON;
my $n = 23;

say encode_json({n => $n});
# {"n":23}
```

```
$n =~ /^\d+$/;

say encode_json({n => $n});
# {"n":"23"}
```

Booleans?

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	boolean.pm

```
use boolean;

my $true  = true;
my $false = false;

say ref $true;
# boolean

say ref $false
# boolean
```

But this is Perl...

TIMTOWTDI

...that's why we can't have nice things

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	boolean.pm

JSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
number	
boolean	\0 \1 boolean.pm JSON::XS::Boolean JSON::PP::Boolean JSON::Tiny::_Bool Mojo::JSON::_Bool Cpanel::JSON::XS::Boolean Types::Serialiser::Boolean ...

Now consider BSON

BSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
boolean	
int32	
int64	
float64 (double)	
decimal128	
binary (blob)	
datetime	
regular expression	
javascript code	
javascript code with data	
"object id"	
"minimum key"	
"maximum key"	
"timestamp"	
(3 deprecated types)	

BSON	Perl
document	hash reference
array	array reference
null	undef
string	Scalar
boolean	
int32	
int64	
float64 (double)	
decimal128	
binary (blob)	
datetime	
regular expression	
javascript code	
javascript code with data	
"object id"	
"minimum key"	
"maximum key"	
"timestamp"	
(3 deprecated types)	Lots of wrapper classes

Serializing Perl is hard

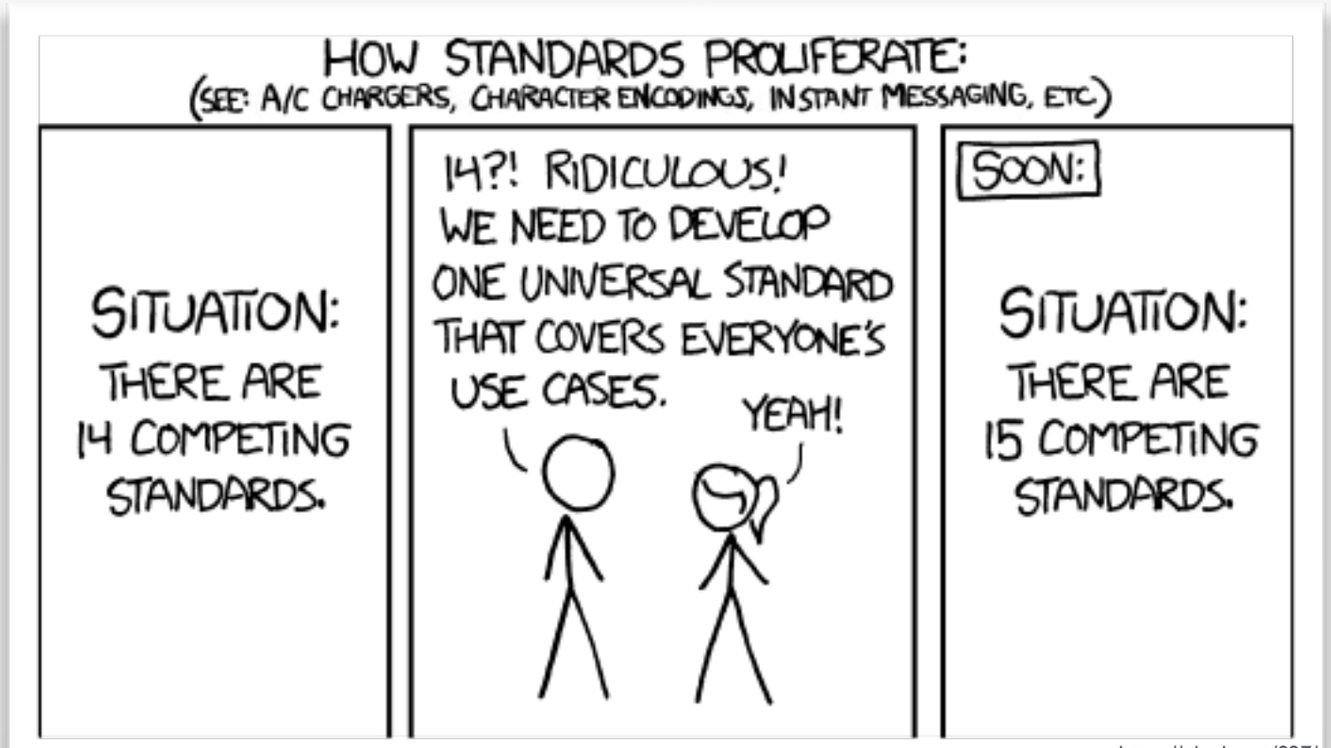
Serializing Perl is hard

Deserializing typed data is hard

1. string/number heuristics

2. wrapper classes

Back to my problem...



https://xkcd.com/927/

MongoDB Perl Driver history and future

MongoDB-v0.x series

BSON codec written in XS

Heuristics configured via global variables

```
($MongoDB::BSON::looks_like_number)
```

Wrapper classes

```
boolean.pm, MongoDB::OID, MongoDB::BSON::Binary,
MongoDB::Code, ...
```

Goals

Provide a pure-Perl option for the Perl driver

Get rid of globals and use codec objects

Allow users to customize decoding to wrappers

Meanwhile on CPAN...

BSON.pm

Mango (for Mojolicious)

Each with their own set of wrapper classes

With generally incompatible APIs

Goals

Provide a pure-Perl option for the Perl driver

Get rid of globals and use codec objects

Allow users to customize decoding to wrappers

Provide standard, reusable BSON wrappers

We adopted BSON.pm

Pure-perl

Not widely used (so we can change it more)

Lots of wrappers (some with odd APIs)

Not tied to Mojolicious ecosystem

Needed a roadmap

Step 1.

Extract MongoDB BSON to a configurable codec object

Define a standard, pluggable codec API

(Released with MongoDB-v1.0.0 in mid-2015)

Step 2.

Adapt BSON.pm to provide the codec API

Adapt or extend wrappers to provide common behaviors needed

Prepare BSON.pm for pluggable backends (e.g. future BSON::XS)

(Released to CPAN as BSON-v1.0.0 in July)

Step 3.

Extract MongoDB XS code to BSON::XS

(Released to CPAN as BSON-XS-v0.2.0 in Oct)

Step 4.

Remove MongoDB codec from distribution

Add dependency on BSON.pm/BSON::XS

Make MongoDB wrapper classes into empty subclasses of BSON.pm wrapper classes

(Planned for future MongoDB-v2.0.0 in 2017)

Hard parts...

Finding common wrapper behavior

BSON LEGACY TYPE SURVEY

NOTE: This document refers to classes from BSON.pm version 0.16, before changes implemented in response to this study. It is preserved in the repository for historical reference only.

Description

This document maps BSON types from the BSON specification to Perl classes that correspond to each type. This document refers to such classes as "BSON type classes" or just "type classes".

The goal of this document is to provide a foundation to determine a way to unify the different BSON type classes. For each type and class, it indicates whether the mapping is one-way or two-way. The public API of the different classes is noted as well.

Broadly speaking, many type classes are drawn from the BSON, MongoDB and Mango Perl distributions. Other CPAN modules serve as type classes in a codec-specific manner.

The BSON types are named according to their libbson terms with their hexadecimal code noted. Additional pseudo-types are noted at the end.

0x01 BSON_TYPE_DOUBLE

`BSON_TYPE_DOUBLE` is serialized and deserialized directly via native Perl floating point (NV types) variables.

No BSON type classes exist specifically for doubles, but see `BSON_TYPE_INT32` and Mango::BSON::Number, which can be used to force double interpretation.

0x02 BSON_TYPE_UTF8

`BSON_TYPE_UTF8` is deserialized to native Perl strings (PV type). Serialization is codec dependent: as Perl scalars may have dual string/numeric representation, codecs may choose to encode scalars as a numeric BSON type or as a `BSON_TYPE_UTF8`.

The following type classes exist to ensure `BSON_TYPE_UTF8` encoding:

MongoDB::BSON::String

14 pages!

https://github.com/mongodb/mongo-perl-bson/blob/master/devel/bson-types-survey.md

```
# MongoDB::OID

my $oid1 = MongoDB::OID->new;
my $oid2 = MongoDB::OID->new(value => $id1->value);
my $oid3 = MongoDB::OID->new($id1->value);
my $oid4 = MongoDB::OID->new($id1);


# BSON::ObjectID

my $oid1 = BSON::ObjectId->new;
my $oid2 = BSON::ObjectId->new($string);
my $oid3 = BSON::ObjectId->new($binary_string);


# Mango::BSON::ObjectID

my $oid1 = Mango::BSON::ObjectID->new;
my $oid2 = Mango::BSON::ObjectID->new('1a2b3c4e5f60718293a4b5c6');
```

Revised or wrote new BSON.pm wrapper classes

Goal: Let all MongoDB::* wrappers become empty subclasses of BSON.pm wrappers

- Lots of constructor argument munging
- Lots of method name aliasing
- Sometimes had to deprecate and start over

Deciding type mapping

Encode

Decode

Scalar

Wrapper A

Wrapper B

Wrapper C

Wrapper D

Wrapper E

Wrapper F

BSON bytes

Wrapper D

Or maybe just scalar based on config?

From BSON.pm docs

Perl type/class ->	BSON type	-> Perl type/class
float[1] BSON::Double	0x01 DOUBLE	float[2]
string[3] BSON::String	0x02 UTF8	string[2]
hashref BSON::Doc BSON::Raw MongoDB::BSON::Raw[d] Tie::IxHash	0x03 DOCUMENT	hashref[4][5]
arrayref	0x04 ARRAY	arrayref
BSON::Bytes scalarref BSON::Binary[d] MongoDB::BSON::Binary[d]	0x05 BINARY	BSON::Bytes
n/a	0x06 UNDEFINED[d]	undef
BSON::OID BSON::ObjectId[d] MongoDB::OID[d]	0x07 OID	BSON::OID
boolean BSON::Bool[d] JSON::XS::Boolean JSON::PP::Boolean JSON::Tiny::_Bool Mojo::JSON::_Bool Cpanel::JSON::XS::Boolean Types::Serialiser::Boolean	0x08 BOOL	boolean
BSON::Time DateTime DateTime::Tiny Time::Moment Mango::BSON::Time	0x09 DATE_TIME	BSON::Time
undef	0x0a NULL	undef
BSON::Regex qr// reference MongoDB::BSON::Regexp[d]	0x0b REGEX	BSON::Regex
n/a	0x0c DBPOINTER[d]	BSON::DBRef
BSON::Code[6] MongoDB::Code[6]	0x0d CODE	BSON::Code
n/a	0x0e SYMBOL[d]	string
BSON::Code[6] MongoDB::Code[6]	0x0f CODEWSCOPE	BSON::Code
integer[7][8] BSON::Int32	0x10 INT32	integer[2]
BSON::Timestamp MongoDB::Timestamp[d]	0x11 TIMESTAMP	BSON::Timestamp
integer[7] BSON::Int64 Math::BigInt Math::Int64	0x12 INT64	integer[2][9]
BSON::MaxKey MongoDB::MaxKey[d]	0x7F MAXKEY	BSON::MaxKey
BSON::MinKey MongoDB::MinKey[d]	0xFF MINKEY	BSON::MinKey

[d] Deprecated or soon to be deprecated.
[1] Scalar with "NV" internal representation no "PV" representation, or a string that looks like a float if the 'prefer_numeric' option is true.
[2] If the 'wrap_numbers' option is true, numeric types will be wrapped as BSON::Double, BSON::Int32 or BSON::Int64 as appropriate to ensure round-tripping. If the 'wrap_strings' option is true, strings will be wrapped as BSON::String, likewise.
[3] Scalar with "PV" representation and not identified as a number by notes [1] or [7].
[4] If 'ordered' option is set, will return a tied hash that preserves order (deprecated 'ixhash' option still works).
[5] If the document appears to contain a DBRef and a 'dbref_callback' exists, that callback is executed with the deserialized document.
[6] Code is serialized as CODE or CODEWSCOPE depending on whether a scope hashref exists in BSON::Code/MongoDB::Code.
[7] Scalar with "IV" internal representation and no "PV" representation, or a string that looks like an integer if the 'prefer_numeric' option is true.
[8] Only if the integer fits in 32 bits.
[9] On 32-bit platforms, 64-bit integers are deserialized to Math::BigInt objects (even if subsequently wrapped into BSON::Int64 if 'wrap_scalars' is true).

Testing PP and XS

Use same tests for both

rsync tests from BSON.pm to BSON::XS

Tests load t/lib/CleanEnv.pm, which hides either BSON::PP (from BSON.pm) or BSON::XS

Ensures both backends consume/produce ***exact*** same BSON

BSON.pm t/lib/CleanEnv.pm

```
use 5.008001;
use strict;
use warnings;

package CleanEnv;

# Tiny equivalent of Devel::Hide to disable BSON::XS
use lib map {
    my ( $m, $c ) = ( $_, qq{die "Can't locate $_ (hidden)\n"} );
    sub { return unless $_[1] eq $m; open my $fh, "<", \$c; return $fh }
} qw{BSON/XS.pm}; # Would be BSON/PP.pm for BSON::XS

# Keep environment from interfering with tests
$ENV{PERL_BSON_BACKEND} = "";

1;
```

Dependency order

Typical Foo/Foo::XS pattern

User uses Foo's API

Foo delegates to XS methods if Foo::XS loads

```
*foo = has_xs() ? \&Foo::XS::_foo : \&Foo::_foo;
```

Foo dynamically adds Foo::XS as a dependency if a compiler is detected

Our problem & solution

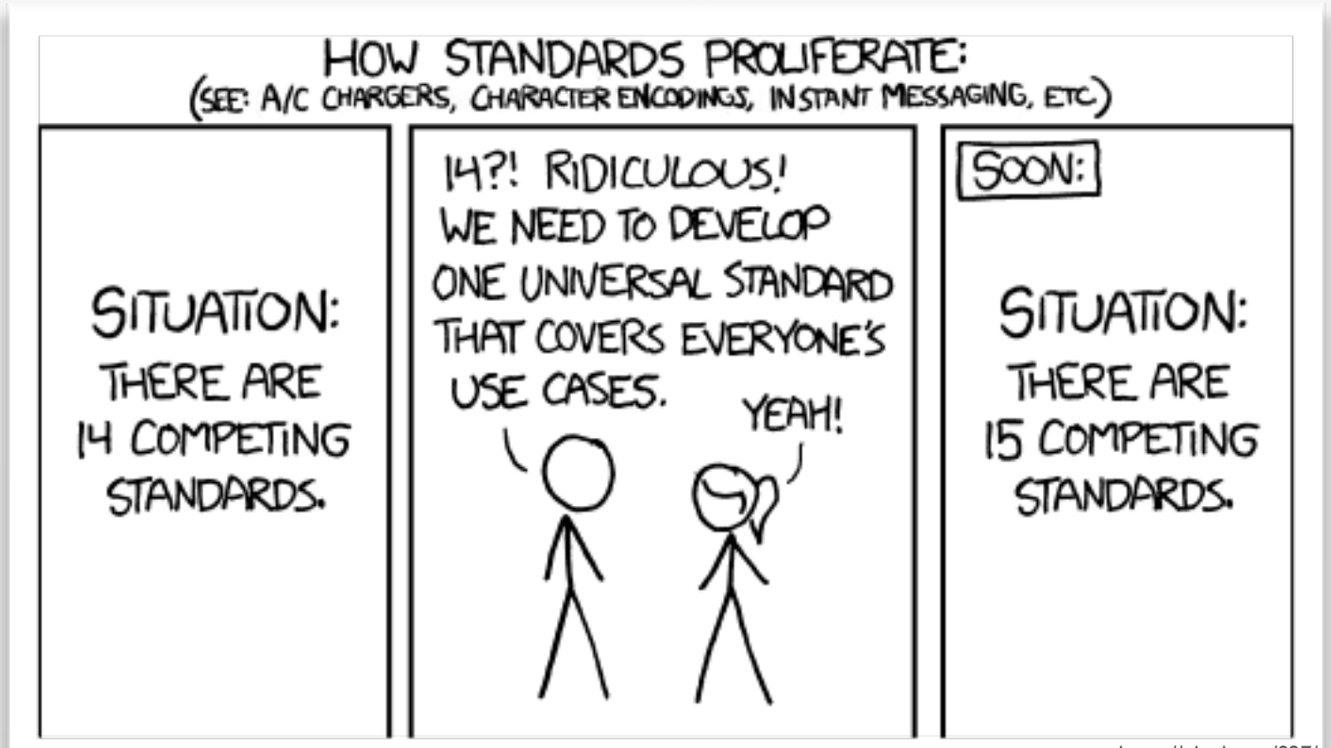
BSON::XS tests need BSON.pm wrappers

Didn't want wrappers as a common prereq

Users can check for a compiler and add BSON.pm or BSON::XS

(Or perhaps I'll just write BSON::MaybeXS)

Standardization is hard



https://xkcd.com/927/

but it can be done

Lessons learned

Serializing needs heuristics and wrapper classes

Type mapping is complex and unfun, but needed

Common tests for PP/XS are a life-saver

Look for MongoDB Perl driver v2.0.0 next year!

Questions?

Email: david@mongodb.com
Twitter/IRC: @xdg