```
----- Forwarded message from Oren Ben-Kiki <orenbk@...> -----
From: Oren Ben-Kiki <orenbk@...>
Subject: RE: Meeting Minutes
Date: Fri, 18 May 2001 19:18:14 +0200

> > 1. Brian stated that he would invstigate Oren's Syntax
> >    and get back with us if it meets Perl's serilization
> >    requirements for hard references.  If not, specify
> >    what alternatives we can use.
>
> I don't think it's that important to investigate. It will probably
> always be a moot point. I will let Data::Denter use it's current scheme
> to deterministically round-trip all Perl data structures. YAML.pm
> probably will have no need for this. It's all acadenic and I have no
> spare time for academics for three more months. (My guess is, yes it
> could be made to work, but would be suboptimal for Perl people) Let's
> leave it at that for now.

Does that mean we are giving up on Denter using YAML syntax (extended to
handle pointer-to-pointer)?

> > 2. We agreed on Oren's reference (&*) syntax.
> >
> > 3. We agreed on having an optiona RFC 822 Header,
> >    this requires that a YAML text without this
> >    header must begin on line #2.  Furthermore,
> >    if an RFC 822 Header is present, then it must
> >    include "X-YAML-Version: 1.0"
> >
> > 4. Brian said that he'd investiage the RFC a bit
> >    more specifically relating to the productions.
> >    (I'm not sure if this is necessary... )

I'm going to go over it with a fine-tooth comb, just to see what is involved
in making YAML a superset of it. I guess I'll also have to look at MIME
while I'm at it, with the same comb :-)

> On 4 & 5. I don't really like the blank line at the beginning thing
> because people will mess it up or not understand it. And we have many
> heuristic options.
>
> A) Parse lookahead for X-YAML-Version
> B) Option-A rarely needed because as soon as we see a key that is *not*
> RFC822 compliant, we assume YAML. 99% of the time this is the first
> line!
> C) If there is no whitespace allowed before the colon in RFC822, we
> simply make it a requirement in YAML. Or does this break your RFC
> compatability rules?
>
> Just for my own edification, would you please explain the rationale
> behind making YAML RFC822 compliant. And do so with one of more specific
> examples. Thanks :)

Well, for example, suppose that YAML was a "good enough" superset of RFC822.
Then we could just adopt my idea that "blank lines separate top-level maps"
and we wouldn't have to say anything further about RFC822 headers, period.
If one wants to read/write a mail message as a YAML document, then it will
simply work (as long as he sticks to the "safe" constructs there). If one
wants to have a YAML document that has nothing to do with RFC822, that also
works. No need for any special statement about them. I like this approach
best.

> > 5. We talked at length about multi-line scalar text
> >    nodes.  Thus far, we agreed on option D, due to
> >    assumed compatibility with RFC 822.  Clark agreed
> >    to verify this compatibility.
> >
> > 6. Brian stated that the quoting mechanism was not
> >    good enough for source code or ASCII art, and
> >    backed option F.  However, option F does not
> >    explicitly preserve trailing whitespace on a
> >    given line.  So Brian suggested using > <
> >    pairs.  Oren suggested using single quotes.
> >    Clark asked Brian to come up with something
> >    he likes and propose it.
>
> Neil and I agree that the normal transport mechanism between Perl and
> Python serializer/parsers would definitely *not* be a mailer. And if a
> mailer was used, most people wouldn't give a darn about the trailing
> whitespace. And if they really did, we could just encode the whole
> whitespace. And if they really did, we could just encode the whole
> document anyway. So I now definitely think the best-fit answer is:
>
>     " this is the hash\n key for this example  :-) " : #class :

I assume the trailing ':' is a typo?

>        |# My Perl Subroutine
>        |

>        |  sub version {
>        |      if ($_[0] =~ /\n/) {
>        |          return \ "\to sender";
>        |      }
>        |  }
>
> Sorry for overloading this example with so many weird things. I'll just
> comment on the multiline semantics:
>
> A) Trailing whitespace is preserved if the transporter preserves it.
> B) The content can always be encoded before transport anyway.
> C) Nothing is escaped. The content is truly verbatim. A '\' is a '\'.
> D) An implicit newline is assumed to be at the end of every line.

We have to decide what our position is about them, BTW. Is a newline a "\n"
or a "\n\r" - the answer may be different in-memory and in the text file
(and thank you, O nameless DOS/CPM programmer, for inflicting this on us :-)

> E) Note that the '|' is one column back from the actual indentation
> level. This is intententional. And it will work even if the indent width
> is set to one character wide. (not mandatory, but I like it.)

Under Python indentation rules, there's no problem indenting the "label"
line by 4 characters and the text lines by 7, or whatever. What you say
about one character indentation, however, implies that the following would
be legal:

    text:
    |multi-line
    |text

I'm not certain I like it. I think Clark should make the call here -
indentation is his baby.

> F) I'd like to push for this always starting on the next line if it is a
> map value. It has no relation to RFC822.

What's the harm in allowing:

    text: | Spaces   and " and \n, oh my!

('"' and '\' meaning themselves in this text).

I don't see why we need to make this distinction between multi- and single-
line text at all. It is bad enough we provide two different quoting
mechanisms...

> > This will work the way I intended it 98% of the time.
>
> > 7. We agreed, after some discussion, that a YAML
> >    parser must support MIME.  We agreed implicity
> >    that it must support base64 encoding.
>
> > 8. We didn't discuss this... but it should be
> >    mentioned that to (a) support unicode and
> >    (b) support RFC 822, our texts must be UTF-8.
> >    Thus a YAML parser/writer will default to

> >    UTF-8, although other encoding support is
> >    optional.

Perhaps we should require that a YAML processor must *accept* UTF-16 files.
That goes well with the "superset" idea. If one wants to write a YAML
document which is also a mail message, he'll just write it in the default
UTF-8 encoding (or at least the first MIME part - I still have to read the
RFC properly).

> > 9. Clark agreed to make a "boostrap" C program
> >    and upload to source forge.  Brian and Neil
> >    agreed to download and hack at will.
>
> As I walked to the train station with Neil, he figured out the C
> implementation in his head and said he would try to get it done before
> bed.
>
> > 10. Oren mentioned that he was thinking of doing
> >     a Java or Javascript implementation.

I started thinking about it and hit on an issue which Brian may already have
thought about - or will have to very soon, if he's covering YAML.pm :-) The
problem is we haven't defined the data model (or, viewing it differently,
the round-tripping issue).

In "dynamic" languages such as Perl, JavaScript, Python (and to some extent,
Java), it is natural to map a YAML map to the native hash, a list to a
vector/array, and a scalar value to a simple string. That works admirably
well, as long as the YAML entity hasn't been annotated with an ID or a class
name.

If one wants to provide a stable-round tripping utility (e.g., suppose I
want to write a YAML pretty printer), where am I to store the ID of a scalar
value? The class of a map? For this use case, it seems my best course of
action is to wrap the native construct (map/list/scalar) in an object which
has an "id", a "class", and a "value".

There are several options:

A) Use the native constructs when possible, and only use "wrapper" objects
when there's a need. That makes access pattern unpredictable: do I write
map{key} or map{key}.value?

B) Always use wrapper objects, and give up on de-serializing YAML into
arbitrary native data structures. Big hit on usefulness - if we do this,
Brian will just give up on us :-)

C) Declare that IDs may be re-written arbitrarily, even by pretty printers.
That is, banish them from the data model.

That leaves "class" as the only problematic issue. We explicitly decided not
to talk about it in the conference call. It seems to me like there's no way
around requiring that this data will survive round-trips, but I also don't
see how it is possible to de-serialize "scalar value" into a normal "Java
String" if someone attached an "unknown" class to it.

So, the idea is to bite the bullet and remove "class" as something specified

in the "label line" (BTW, we need to define some terminology here; I'm using
"label lines" and "text lines" - or maybe it should be "content lines"?). It
turns out that we can still achive most of the goals of the "class"
construct by making the key "#" magical in maps:

    center: %
        #: point
        x: 35.3
        y: 42.1

The idea is that the "point" class knows it has members "x" and "y" and that
their values must be floating-point numbers; so not being to declare their
type is acceptable.

When serializing this to a language/system which doesn't recognize "point",
well, "center" will just be a run-of-the-mill map. The only magic here is
that we may require '#' to always be printed first, but this depends on the
protocol we'll be using for constructing "point" objects when the class is
recognized:

C1) If the interface is such that there has to be a factory method accepting
a map of all the keys, and returning the constructed object, than this
restriction is unnecessary. This is a good way to do things in
Perl/JavaScript object creation interfaces; e.g. in Perl the method will
typically merely bless the map and be done. I suspect in Python things would
be a bit less elegant.

C2) In Java this may be less efficient (all these map and String objects
will have to be created and destroyed per each point object creation -
slooow!). Any more efficient method will have to involve tighter interaction
between parsing the values and creating the object, which means we have to
know its class when starting to parse each member (e.g., we may be able to
parse "42.1" directly into a float). I can't see, off the top of my head, a
reasonable protocol for this right now, but we may want to require '#' to be
the first key anyway, in case something does come up.

My favorite is C2. It's down side is that you can't directly assign a type
for a list or a scalar; you have to assign it to a surrounding map. I forsee
a long discussion coming...
```

```
> > 11. Clark agreed to update the spec with the
> >     current agreements.
>
> Send me a note when it's changed. I'll review for you.
>
> >
> > 12. Brian mentioned that he'd show YAML to one of
> >     his Perl friends.  (sorry I didn't catch his name)
>
> Damian Conway http://www.csse.monash.edu.au/~damian/

His input will be greatly appreciated.

> > 13. Clark and Brian discussed the MIME usage.

And...?


> > 14. Clark introduced a very short discussion on
> >     the need for a global mechanism to uniquely
> >     identify names in a non-language specific
> >     manner.

Reverse DNS being the easiest way we've ever come up with for this. It works
directly for accessing class names in Java. In C++ you'll have to "register"
classes manually anyway, so using reverse DNS doesn't gain or cost anything.
In Perl/JavaScript/Python we may want to set up some automatic way to
convert "org.yaml.class" into something the native type system recognizes...

> > 15. Clark agreed to write up the "single vs multi"
> >     line controversy and post to the list so that
> >     it is clearly understood.

I thought we settled this... Every scalar value is potentially multi-line.
It doesn't seem to cost us anything, or does it?

> > 16. We made little progress on the scalar indicator
> >     for lists, to colon or not to colon.  It wasn't
> >     agreed, but Clark thinks this is someone else's
> >     monkey.  If Oren and Brian can't agree within
> >     7 days, Clark will put on the dictator cap.
>
> We traded in the '$' for the ':'. '$' as the last character in a line

I thought ':' was the first one; it is "as if" it is a normal header, with
the key "just happening" to be empty. This seems more consistent.

> meant a multiline scalar was to follow. Converting this semantic to the
> ':' leaves us with these represntations:
>
>     key1 : @
>         single line
>         :
>             classless folded
>             multi line
>         another single line
>         and another
>         #class &0001 :

          : #class &0001

>             classed multi
>             line
>         #class &0002 classed single line
>         %
>             key : value
>         @

This is an empty list, right?

>         ~

And this is a null?

>         #classy %

>             key : value
>         : even this multi line on the same line
>           as a colon thingy works because there
>           a little bit of indentation imposed by
>           colon. (Although I don't love it)
```

```
This means the following:

          : single line

Will also work, even though you *really* dislike it. I like them :-)

>         : "Another thingy like above that meets"
>           "RFC822 wackiness"
>         :
>            |    1
>            |   1 1
>            |  1 1 1
>            |Just for completeness :-)


I think we've said everything there's to be said about this, and whether or
not you find either:

    list:
      : One
      : Two
      : Three
        and Four

Or:

    list:
        One
        Two
        :
            Three
            and Four

To be beautiful or ugly is, when all is said and done, a matter of taste. To
you, the extra ':'s are an eyesore; to me it seems strange that the
multi-line value is "more indented"; it seems as though there's structure
involved, when there isn't. I also like being able to do /^:/ in VI to get
to the next entry.

I think we should just let Clark make the call. OK?

> > 17. It is nice to have Neil in on the talk!

Welcome aboard, Neil. The more the merrier!

Have fun,

    Oren Ben-Kiki

----- End forwarded message -----
```

Re: [Yaml-core] (backfill) YAML Meeting: Oren's Feedback

From: Clark C . Evans <cce@cl...> - 2001-05-18 20:32

```
| > Sorry for overloading this example with so many weird things. I'll just
| > comment on the multiline semantics:
| >
| > A) Trailing whitespace is preserved if the transporter preserves it.
| > B) The content can always be encoded before transport anyway.
| > C) Nothing is escaped. The content is truly verbatim. A '\' is a '\'.
| > D) An implicit newline is assumed to be at the end of every line.
|
| We have to decide what our position is about them, BTW. Is a newline a "\n"
| or a "\n\r" - the answer may be different in-memory and in the text file
| (and thank you, O nameless DOS/CPM programmer, for inflicting this on us :-)

FYI, XML chose the option to convert, at parse time, "\n\r"
and "\r" to "\n".  In this way, anyone writing XML tools
could use \n and not worry about the platform's specific
line ending.  XML does not specify how these must be written
```

```
out, so that any of the three common line endings can be
used on serilization depending on the platform.


| > E) Note that the '|' is one column back from the actual indentation
| > level. This is intententional. And it will work even if the indent width
| > is set to one character wide. (not mandatory, but I like it.)
|
| Under Python indentation rules, there's no problem indenting the "label"
| line by 4 characters and the text lines by 7, or whatever. What you say
| about one character indentation, however, implies that the following would
| be legal:
|
|     text:
|     |multi-line
|     |text
|
| I'm not certain I like it. I think Clark should make the call here -
| indentation is his baby.

How about we define indentation as leading whitespace?  In this
case, the minimal indentation for this style would be two.

  text:
   |this is my
   |multi-line

|
| > F) I'd like to push for this always starting on the next line if it is a
| > map value. It has no relation to RFC822.
|
| What's the harm in allowing:
|
|     text: | Spaces   and " and \n, oh my!
|
| ('"' and '\' meaning themselves in this text).
|
| I don't see why we need to make this distinction between multi- and single-
| line text at all. It is bad enough we provide two different quoting
| mechanisms...

Beacuse then we have to answer questions like...

  text: |does X align vertically?
   |          X

I like Brian's restriction on this form.


| Perhaps we should require that a YAML processor must *accept* UTF-16 files.
| That goes well with the "superset" idea. If one wants to write a YAML
| document which is also a mail message, he'll just write it in the default
| UTF-8 encoding (or at least the first MIME part - I still have to read the
| RFC properly).


So we must accept both UTF-8 and UTF-16.  Where UTF-16
support then implies that the RFC 822 headers are not present?
Or, do we upgrade the spec and allow the headers as
long as the BOM is there?

| > > 10. Oren mentioned that he was thinking of doing
| > >     a Java or Javascript implementation.
|
| I started thinking about it and hit on an issue which Brian may already have
| thought about - or will have to very soon, if he's covering YAML.pm :-) The
| problem is we haven't defined the data model (or, viewing it differently,
| the round-tripping issue).
|
| In "dynamic" languages such as Perl, JavaScript, Python (and to some extent,
| Java), it is natural to map a YAML map to the native hash, a list to a
| vector/array, and a scalar value to a simple string. That works admirably
| well, as long as the YAML entity hasn't been annotated with an ID or a class
| name.

We will officially decree that the ID is NON INFORMATIONAL.
We must be very careful about this ID.  It smells alot like
an XML prefix tar baby... which completely destroyed any
hopes at an XML canonical form.

| If one wants to provide a stable-round tripping utility (e.g., suppose I
| want to write a YAML pretty printer), where am I to store the ID of a scalar
```

| value? The class of a map? For this use case, it seems my best course of
| action is to wrap the native construct (map/list/scalar) in an object which
| has an "id", a "class", and a "value".

I must say, I don't like this option.

| A) Use the native constructs when possible, and only use "wrapper" objects
| when there's a need. That makes access pattern unpredictable: do I write
| map{key} or map{key}.value?
|
| B) Always use wrapper objects, and give up on de-serializing YAML into
| arbitrary native data structures. Big hit on usefulness - if we do this,
| Brian will just give up on us :-)
|
| C) Declare that IDs may be re-written arbitrarily, even by pretty printers.
| That is, banish them from the data model.

Yes. Option C. For the sequential API, we will have
to expose this, but not in the in-memory representation.
And in the sequential API, it can be wrapped as an
opaque handle (without a string representation).
Under no circumstances do we want this ID to become
"data" as the XML prefix has... less we not have a
reasonable canonical form.

Also, to enable concatination, I think we should
allow IDs to be shadowed. In other words,

```
  a : &0001  "This is a value"
  b : *0001
  c : &0001  "This is another value"
  d : *0001
```

In this case, d->c and b->a. After c, there is no
way to access a by reference. Simple solution,
and this way concatination is still well defined.

| That leaves "class" as the only problematic issue. We explicitly
| decided not to talk about it in the conference call. It seems to
| me like there's no way around requiring that this data will survive
| round-trips, but I also don't see how it is possible to de-serialize
| "scalar value" into a normal "Java String" if someone attached an
| "unknown" class to it.

If classes arn't available in the target environment,
or if the class requested can't be found, then we
have a slight problem. A resonable solution is
to notify the user via a warning, and then create
an auxilary yaml-class-map which maps lists/strings/maps
that have been created by the load with their
corresponding class name. In this way we keep the
native structures, but preserve the class names through
a "coloring archive" on the side.

| So, the idea is to bite the bullet and remove "class"
| as something specified in the "label line" (BTW, we need
| to define some terminology here; I'm using "label lines"
| and "text lines" - or maybe it should be "content lines"?).
| It turns out that we can still achive most of the goals
| of the "class" construct by making the key "#" magical in maps:

```
|
|     center: %
|         #: point
|         x: 35.3
|         y: 42.1
```

Interesting. However, this prevents class names for
Strings or Lists. Very interesting. What do we do about
Strings and Lists? Move this into the category of "non-portable"
constructs, like a & and * on the same line? I'm not sure.
The "coloring on the side" may be more painful (esp garbage
collecting), but it at least does not get in the way. Hmm.

| The idea is that the "point" class knows it has members "x"
| and "y" and that their values must be floating-point numbers;
| so not being to declare their type is acceptable.
|
| When serializing this to a language/system which doesn't
| recognize "point", well, "center" will just be a run-of-the-mill
| map. The only magic here is that we may require '#' to always
| be printed first, but this depends on the protocol we'll be using

```
| for constructing "point" objects when the class is recognized:
|
| C1) If the interface is such that there has to be a factory
| method accepting a map of all the keys, and returning the
| constructed object, than this restriction is unnecessary. This
| is a good way to do things in Perl/JavaScript object creation
| interfaces; e.g. in Perl the method will typically merely bless
| the map and be done. I suspect in Python things would
| be a bit less elegant.

Python is very flexible here...

| C2) In Java this may be less efficient (all these map and
| String objects will have to be created and destroyed per each
| point object creation - slooow!). Any more efficient method will
| have to involve tighter interaction between parsing the values
| and creating the object, which means we have to know its class
| when starting to parse each member (e.g., we may be able to
| parse "42.1" directly into a float). I can't see, off the top of
| my head, a reasonable protocol for this right now, but we may
| want to require '#' to be the first key anyway, in case something
| does come up.
|
| My favorite is C2. It's down side is that you can't directly assign
| a type for a list or a scalar; you have to assign it to a
| surrounding map. I forsee a long discussion coming...

And we have not even begun to talk about namespaces, or how
to make a name globally unique so that it can be moved across
multiple languages.  This is especially important for common
objects, like Date, Currency, and possibly even Party or Address.

| > > 14. Clark introduced a very short discussion on
| > >     the need for a global mechanism to uniquely
| > >     identify names in a non-language specific
| > >     manner.

|
| Reverse DNS being the easiest way we've ever come up
| with for this. It works directly for accessing class names
| in Java. In C++ you'll have to "register" classes manually
| anyway, so using reverse DNS doesn't gain or cost anything.
| In Perl/JavaScript/Python we may want to set up some automatic
| way to convert "org.yaml.class" into something the native
| type system recognizes...

Python has a package mechanism similar to Java.

| > > 15. Clark agreed to write up the "single vs multi"
| > >     line controversy and post to the list so that
| > >     it is clearly understood.
|
| I thought we settled this... Every scalar value is potentially multi-line.
| It doesn't seem to cost us anything, or does it?

Good.  I'll just document it in the updated spec... coming soon.

Clark
```

Re: [Yaml-core] (backfill) YAML Meeting: Oren's Feedback

From: Brian Ingerson <briani@Ac...> - 2001-05-18 23:48

```
"Clark C . Evans" wrote:
> How about we define indentation as leading whitespace?  In this
> case, the minimal indentation for this style would be two.
>
>   text:
>    |this is my
>    |multi-line
>

Hopefully, you saw my last comment on this. I now think the '|' should
```

```
be in the first column *after* indentation.

> |
> | > F) I'd like to push for this always starting on the next line if it is a
> | > map value. It has no relation to RFC822.
> |
> | What's the harm in allowing:
> |
> |     text: | Spaces   and " and \n, oh my!
> |
> | ('"' and '\' meaning themselves in this text).
> |
> | I don't see why we need to make this distinction between multi- and single-
> | line text at all. It is bad enough we provide two different quoting
> | mechanisms...
>
> Beacuse then we have to answer questions like...
>
>   text: |does X align vertically?
>    |          X
>
> I like Brian's restriction on this form.

Actually there's no restriction. It would look like this:

   text: |does X align vertically?
         |          X

But this would *not* be the default.

> Yes. Option C.  For the sequential API, we will have
> to expose this, but not in the in-memory representation.
> And in the sequential API, it can be wrapped as an
> opaque handle (without a string representation).
> Under no circumstances do we want this ID to become
> "data" as the XML prefix has... less we not have a
> reasonable canonical form.
>
> Also, to enable concatination, I think we should
> allow IDs to be shadowed.  In other words,
>
>   a : &0001  "This is a value"
>   b : *0001
>   c : &0001  "This is another value"
>   d : *0001
>
> In this case, d->c and b->a.  After c, there is no
> way to access a by reference.  Simple solution,
> and this way concatination is still well defined.
>

I support this.

> | That leaves "class" as the only problematic issue. We explicitly
> | decided not to talk about it in the conference call. It seems to
> | me like there's no way around requiring that this data will survive
> | round-trips, but I also don't see how it is possible to de-serialize
> | "scalar value" into a normal "Java String" if someone attached an
> | "unknown" class to it.
>
> If classes arn't available in the target environment,
> or if the class requested can't be found, then we
> have a slight problem.  A resonable solution is
> to notify the user via a warning, and then create
> an auxilary yaml-class-map which maps lists/strings/maps
> that have been created by the load with their
> corresponding class name.  In this way we keep the
> native structures, but preserve the class names through
> a "coloring archive" on the side.
>

> | So, the idea is to bite the bullet and remove "class"
> | as something specified in the "label line" (BTW, we need
> | to define some terminology here; I'm using "label lines"
> | and "text lines" - or maybe it should be "content lines"?).
> | It turns out that we can still achive most of the goals
> | of the "class" construct by making the key "#" magical in maps:
> |
> |     center: %
> |         #: point
> |         x: 35.3
> |         y: 42.1
```

```
>
> Interesting.  However, this prevents class names for
> Strings or Lists.  Very interesting.  What do we do about
> Strings and Lists?  Move this into the category of "non-portable"
> constructs, like a & and * on the same line?  I'm not sure.
> The "coloring on the side" may be more painful (esp garbage
> collecting), but it at least does not get in the way.  Hmm.

I can live without class names for Scalars and Arrays, but Hashes are a
must. Most perl objects map into Hashes. Even Array and Scalar ones can
be serialized into Hashes. Hashes work for all Python objects and
Javascript too, AFAIK. I'll bet they could work for Java and C++ as
well.

Now we could make a distinction between Objects and Hashes. Just have
Objects be separate regardless of their implementation. Just pick
another symbol. '&' is available if we go back to %(0001) notation.

Don't leave me without an object model or I'll go Postal %\


Still smiling, Brian

--
perl -le 'use Inline C=>q{SV*JAxH(char*x){return newSVpvf
("Just Another %s Hacker",x);}};print JAxH+Perl'
```