Home / Browse / YAML Ain't Markup Language / Mail / Archive

YAML Ain't Markup Language

Summary Files Reviews Support Develop Tracker Mailing Lists Forums Code

Email Archive: yaml-core (read-only)

2001:	Jan	Feb	Mar	Apr	May (101)	Jun (157)	Jul (89)	Aug (135)	Sep (17)	Oct (86)	Nov (410)	Dec (311)
2002:	Jan (76)	Feb (100)	Mar (139)	Apr (138)	May (234)	Jun (178)	Jul (271)	Aug (286)	Sep (816)	Oct (50)	Nov (28)	Dec (137)
2003:	Jan (62)	Feb (25)	Mar (97)	Apr (34)	May (35)	Jun (32)	Jul (32)	Aug (57)	Sep (67)	Oct (176)	Nov (36)	Dec (37)
2004:	Jan (20)	Feb (93)	Mar (16)	Apr (36)	May (59)	Jun (48)	Jul (20)	Aug (154)	Sep (868)	Oct (41)	Nov (63)	Dec (60)
2005:	Jan (59)	Feb (15)	Mar (16)	Apr (14)	May (19)	Jun (16)	Jul (25)	Aug (19)	Sep (7)	Oct (12)	Nov (18)	Dec (41)
2006:	Jan (16)	Feb (65)	Mar (51)	Apr (75)	May (38)	Jun (25)	Jul (23)	Aug (16)	Sep (24)	Oct (3)	Nov (1)	Dec (10)
2007:	Jan (4)	Feb (5)	Mar (7)	Apr (29)	May (38)	Jun (3)	Jul (1)	Aug (17)	Sep (1)	Oct	Nov (11)	Dec (16)
2008:	Jan (11)	Feb (4)	Mar (7)	Apr (48)	May (17)	Jun (9)	Jul (6)	Aug (12)	Sep (5)	Oct (7)	Nov (4)	Dec (11)
2009:	Jan (15)	Feb (28)	Mar (12)	Apr (44)	May (6)	Jun (16)	Jul (6)	Aug (37)	Sep (107)	Oct (24)	Nov (30)	Dec (22)
2010:	Jan (8)	Feb (16)	Mar (11)	Apr (28)	May (9)	Jun (26)	Jul (7)	Aug (25)	Sep (2)	Oct	Nov	Dec
2011:	Jan (5)	Feb (6)	Mar (3)	Apr (2)	May (10)	Jun (44)	Jul (11)	Aug (8)	Sep (6)	Oct (42)	Nov (19)	Dec (5)
2012:	Jan (23)	Feb (8)	Mar (9)	Apr (11)	May (2)	Jun (11)	Jul	Aug (18)	Sep (1)	Oct (15)	Nov (14)	Dec (8)
2013:	Jan (5)	Feb (13)	Mar (2)	Apr (10)	May	Jun (6)	Jul (17)	Aug (2)	Sep (2)	Oct	Nov	Dec

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-05-31 11:45

```
> I don't see why we need to forbid it's usage in lists...
> although you may consider it ugly...
>
>   list: @
>     This is a multi line
>      scalar value.
>     This is the second entry.

First, it is specifically forbidden by the current spec.
Second, yes, it is ugly, a prefix ':' would do wonders here. I've promised
not to ask again for allowing it, so I won't :-)

> | I don't like it because:
> |
> | - There's no way to write Unicode blocks.
> | - You are mixing up ASCII text and binary data in the same
> | data type,
> | - And separating ASCII and Unicode text for no good reason.
> | Languages are
> | evolving towards "text = Unicode", as they should, and
> | UTF-8 makes it very
> | easy to deal with Unicode text even in languages such as C.
>
> Yep.  I can see these problems.  However, then our YAR
> use case must have a ".yar" file which includes the list
> of "text" extensions.  Furthermore, it will have to also
> note if the file was stored as UTF-8 or UTF-16.  Hmmm.

Let's see. YAR needs to satisfy the following requirements:

1. Always round-trip the file correctly, byte-to-byte identical.
2. Subject to (1), use a human-readable representation of a file.
3. Carry over any meta-data about the file which is relevant.

Solution:

- Represent each file as a map:
    file-name: %
        permissions: ...
        owner: ...
        group: ...
        character-set: ...
        content: ...

- The "character set" attribute explains how to convert the value of the
'content' entry to bytes to write. It should be one of
ascii/utf-8/utf-16be/utf-16le. If the content is a binary blob, either there
is no "character set" (the content isn't textual), or it could just say
"binary" (I know that's not a IANA recognized character set...)

- The yar file creator chooses which syntax format to give the content
according to the following heuristic. In each case, it ensures the encoding
will be such that the file will be re-created byte-to-byte identical to the
original:

If the file contains...
1. ... only 7-bit printable ASCII characters, plus newlines => syntax:
block, charset: ascii.
2. ... only utf-8 printable characters, plus newlines => syntax: block,
charset: utf-8.
3. ... like 1, with the odd 8-bit character (but not 2) => syntax: quoted
string, charset: ascii.
4. ... like 2, with the odd unprintable character => syntax: quoted string,
charset: utf-8.
5. ... what looks to be as a utf-16 file with only printable characters plus
newlines => syntax: block charset: utf-16be/utf-16le.
6. ... like 5 with the odd non-printable character => syntax: quoted string,
charset: utf-16be/utf-16le.
7. ... anything else => syntax: blob, charset: binary (or missing).

This works, is safe, is extensible, and doesn't require us to bend YAML out
of shape. The only things which bothers me is handling newlines when
transferring between DOS/Windows and the rest of the world. Question: in a
block, does YAML preserve whether a line ended with a \n or a \r\n? If not
we are OK.

And I'm still worried about the class marker... Can you say something about
what you mean by "class map"? I didn't get it.

Have fun
```

```
have fun,

    Oren Ben-Kiki
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-05-31 11:51

```
I meant to write:

> Question: in a
> block, does YAML preserve whether a line ended with a \n or a
> \r\n? If not
^^^^^^^^^ yes
> we are OK.

Oren.
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-05-31 16:34

```
yOn Thu, May 31, 2001 at 01:46:01PM +0200, Oren Ben-Kiki wrote:
| > I don't see why we need to forbid it's usage in lists...
| > although you may consider it ugly...
| >
| >   list: @
| >     This is a multi line
| >      scalar value.
| >     This is the second entry.
|
| First, it is specifically forbidden by the current spec.

It is?  It was in a previous spec, but not the recent
one (the one dated the 26th) which was updated in a major
way on the 26th after it was released (my bad).
Where explicitly, as I actually remember working this
through in my head so that it wasn't an exception.

| Second, yes, it is ugly, a prefix ':' would do wonders here.

Hmm.  Considering.


| Let's see. YAR needs to satisfy the following requirements:
|
| 1. Always round-trip the file correctly, byte-to-byte identical.
| 2. Subject to (1), use a human-readable representation of a file.
| 3. Carry over any meta-data about the file which is relevant.

Good.


| - Represent each file as a map:
|     file-name: %
|         permissions: ...
|         owner: ...
|         group: ...
|         character-set: ...
|         content: ...
|
| - The "character set" attribute explains how to convert the value of the
| 'content' entry to bytes to write. It should be one of
| ascii/utf-8/utf-16be/utf-16le. If the content is a binary blob, either there
| is no "character set" (the content isn't textual), or it could just say
| "binary" (I know that's not a IANA recognized character set...)

Ok.  So we've made an explicit node attribute
called "character set".  Interesting.  Should
this be added using a special indicator?  ^utf-8

| This works, is safe, is extensible, and doesn't require us to bend YAML out
| of shape. The only things which bothers me is handling newlines when
| transferring between DOS/Windows and the rest of the world. Question: in a
| block, does YAML preserve whether a line ended with a \n or a \r\n? If not
| we are OK.

Yep.  I was worring about this one too.  I'm not
sure what the solution is.  Normalizing new lines
is a perfectly reasonable thing to do... except
in this case.

| And I'm still worried about the class marker... Can you
| say something about what you mean by "class map"? I
| didn't get it.

Assume you only have a object which is a map, list, or scalar.
The YAML parser/emitter pair could keep a global "classmapping"
which associated all objects created via reading from the
serialized format with it's optional class map.  Then, when
writing the YAML file, the global variable could be used to
reconstruct the class map attribute.  Yes, this opens up
a small can of worms... garbage collection among them.
However, it does make round-tripping possible even when
class isn't supported by the system.

Best,

Clark
```

Re: [Yaml-core] yar

From: Brian Ingerson <briani@Ac...> - 2001-05-31 18:35

```
Oren Ben-Kiki wrote:
>
> I meant to write:
>
> > Question: in a
> > block, does YAML preserve whether a line ended with a \n or a
> > \r\n? If not
> ^^^^^^^^^ yes
> > we are OK.

Then by all means, preserve.

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

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-01 05:18

```
On Thu, May 31, 2001 at 01:46:01PM +0200, Oren Ben-Kiki wrote:
| - The "character set" attribute explains how to convert the value of the
| 'content' entry to bytes to write. It should be one of
| ascii/utf-8/utf-16be/utf-16le. If the content is a binary blob, either there
| is no "character set" (the content isn't textual), or it could just say
| "binary" (I know that's not a IANA recognized character set...)

I was thinking a bit differently.  Perhaps either
we should use "class" to indicate character set
or introduce another indicator.  Your thoughts.
Consider the current "built-in" classes,
"_int", "_real", etc.   Are not they encodings?

| The only things which bothers me is handling newlines when
| transferring between DOS/Windows and the rest of the world.

This was bugging me, but I like the YAML spec as it is.

| Question: in a block, does YAML preserve whether a line
| ended with a \n or a \r\n?

No.  This is one of the nice problems XML cleared up, let's
not roll back the clock.  I've yet to hear anyone complain
that XML's folding of \r\n and \r into just \n was anything
but helpful.

The only place it gets us into problems is for the YAR
use case... where the serializer doesn't know if a given
value is character or binary.   In this case, I think
the behavior should be dependent upon the platform.
On unix platforms, files with \r\n are treated as binary
(to preserve the line endings), and on DOS boxes, files
ending with \n (without \r) are treated as binary.

This gives the expected behavior when moving files
between platforms.  By the way, with the current
treatment, any YAML saved on a Windows box will
have \r\n line endings.  And likewise, on a
Unix box it will have \n line endings.  Since the
parser doesn't care, one can move the files back
and forth without affecting the canonical form.

This may not give exactly the same behavior
as "tar", since the exact line endings won't
be preserved when switching platforms... but this
is always a sore spot anyway!  As for "diff",
there are (or should be) flags to not report
line ending differences.

Therefore, I'm pretty certain that I'd like to keep
the line ending folding as it is in the YAML spec.

Best,

Clark
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-03 06:54

```
Clark C . Evans [mailto:cce@...] wrote:
> Ok.  So we've made an explicit node attribute
> called "character set".  Interesting.  Should
> this be added using a special indicator?  ^utf-8

Eeek. What next, a special notation for permissions (!), owner (~), and
group (?)?

I thought YAML was to use the minimal number of special properties - data
type, and reference handling, and (maybe!) classes. Anything else can and
should be done by using normal map keys.

> | And I'm still worried about the class marker... Can you
> | say something about what you mean by "class map"? I
> | didn't get it.
>
> Assume you only have a object which is a map, list, or scalar.
> The YAML parser/emitter pair could keep a global "classmapping"
> which associated all objects created via reading from the
> serialized format with it's optional class map.  Then, when
> writing the YAML file, the global variable could be used to
> reconstruct the class map attribute.  Yes, this opens up
> a small can of worms... garbage collection among them.
> However, it does make round-tripping possible even when
> class isn't supported by the system.

In essence you are proposing an alternate color mechanism - one based on a
separate store, "to the side of" the document, which contains the "color"
info for each node. In other words, admitting that YAML isn't good enough by
itself to handle this data and therefore a separate mechanism must be added
to it. I strongly dislike this.

Just like YAR is a great, concrete example of an application for resolving
syntax issues etc. I feel that classes are a great, concrete example of an
application for resolving the color issue. Let's "eat our own dog food"
here.

Have fun,

    Oren Ben-Kiki
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-03 07:11

```
Clark C . Evans [mailto:cce@...] wrote:
> On Thu, May 31, 2001 at 01:46:01PM +0200, Oren Ben-Kiki wrote:
> | - The "character set" attribute explains how to convert the
> | value of the
> | 'content' entry to bytes to write. It should be one of
> | ascii/utf-8/utf-16be/utf-16le. If the content is a binary
> | blob, either there
> | is no "character set" (the content isn't textual), or it
> | could just say
> | "binary" (I know that's not a IANA recognized character set...)
>
> I was thinking a bit differently.  Perhaps either
> we should use "class" to indicate character set
> or introduce another indicator.  Your thoughts.
> Consider the current "built-in" classes,
> "_int", "_real", etc.   Are not they encodings?

It seems I'm making a distinction here that you aren't.

Issue 1: How to convert YAML text to an in-memory object and vice-versa
(YAML issue).
Issue 2: How to convert YAR file content represented in a YAML scalar to a
file on the disk and vice versa (YAR issue).

Character set, as I described it, belongs to issue 2. It seems to me
completely outside the scope of YAML to include, as part of the object
model, the answer to the question "if this Unicode string was to be written
as the sole content of an on-disk file, which of the many possible encodings
to use for it".

Your questions about the built-in types, however, relate to issue 1: When
reading the YAML file into memory in, say, Java, should I create a String
in-memory object, a Float in-memory object (or maybe a Double?), or an
Integer in-memory object (or maybe a Long?). Great question, to which my
answer is: "determine how YAML will support color, and do it that way".

> | Question: in a block, does YAML preserve whether a line
> | ended with a \n or a \r\n?
>
> No.  This is one of the nice problems XML cleared up, let's
> not roll back the clock.  I've yet to hear anyone complain
> that XML's folding of \r\n and \r into just \n was anything
> but helpful.
>
> The only place it gets us into problems is for the YAR
> use case... where the serializer doesn't know if a given
> value is character or binary.   In this case, I think
> the behavior should be dependent upon the platform.
> On unix platforms, files with \r\n are treated as binary
> (to preserve the line endings), and on DOS boxes, files
> ending with \n (without \r) are treated as binary.

A "binary" file will be represented as a base64 blob in the YAR file. You
don't have to go that far to handle an occasional \r\n or \n in a file; you
can just use a quoted string and escape them properly. That's still "mostly
text".

> This gives the expected behavior when moving files
> between platforms.  By the way, with the current
> treatment, any YAML saved on a Windows box will
> have \r\n line endings.  And likewise, on a
> Unix box it will have \n line endings.  Since the
> parser doesn't care, one can move the files back
> and forth without affecting the canonical form.

That's neat.

> This may not give exactly the same behavior
> as "tar", since the exact line endings won't
> be preserved when switching platforms... but this
> is always a sore spot anyway!  As for "diff",
> there are (or should be) flags to not report
> line ending differences.

So, YAR is more like "shar" then "tar". That makes a lot of sense.

> Therefore, I'm pretty certain that I'd like to keep
> the line ending folding as it is in the YAML spec.

OK.

Have fun,

    Oren Ben-Kiki
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-03 07:58

```
Clark C . Evans [mailto:cce@...] wrote:
> yOn Thu, May 31, 2001 at 01:46:01PM +0200, Oren Ben-Kiki wrote:
> | > I don't see why we need to forbid it's usage in lists...
> | > although you may consider it ugly...
> | >
> | >   list: @
> | >     This is a multi line
> | >      scalar value.
> | >     This is the second entry.
> |
> | First, it is specifically forbidden by the current spec.
>
> It is?  It was in a previous spec, but not the recent
> one (the one dated the 26th) which was updated in a major
> way on the 26th after it was released (my bad).
> Where explicitly, as I actually remember working this
> through in my head so that it wasn't an exception.

I just checked and you are right - I have a printed version of the "older"
26th draft. Sorry.

Things about the latest one which we agreed to fix:

- Indentation in quoted strings;
- Separate text from binary in the information model;
- Clearing up how the API handles "multi-map" files (with blank lines).

Other things which I noticed:

- The draft requires a blank line at the end of the document;
- There are some formatting issues;
- There's a special standing for ISO=8859-1 which I don't follow. Isn't
everything in Unicode? What does it mean to write an ISO-8859-1 escape
within a quoted string?

And, there's the class issue...

Have fun,

    Oren Ben-Kiki
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-03 13:46

```
> Things about the latest one which we agreed to fix:
>
> - Indentation in quoted strings;
And blobs...
> - Separate text from binary in the information model;
> - Clearing up how the API handles "multi-map" files (with
> blank lines).
>
> Other things which I noticed:
>
> - The draft requires a blank line at the end of the document;
> - There are some formatting issues;
> - There's a special standing for ISO=8859-1 which I don't
> follow. Isn't
> everything in Unicode? What does it mean to write an ISO-8859-1 escape
> within a quoted string?
>
> And, there's the class issue...

Have fun,

    Oren Ben-Kiki
```

RE: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-03 15:47

```
On Sun, Jun 03, 2001 at 05:15:56PM +0200, Oren Ben-Kiki wrote:
| I had this notion about the syntactical form of class-as-color (or any
| color). What if one allows the first key pair to appear immediately after
| the '%':
|
|     delivery: % : 10-JAN-2001
|         # : date
|
| OK, the class has to appear on a separate line, but that's not *too* bad,
| textually. Especially after removing some whitespace:
|
|     delivery:%: 10-JAN-2001
|         #: date

If we want to "abbreviate", why not just allow
the single line form...

  delivery: !date 10-JAN-2001

to be an abbreviation for ...

  delivery: %
     !  : date
```

```
      "" : 10-JAN-2001


  Thoughts?  The above form is what would be
  exposed through the information model.

  | I'm using your "value is the empty key" syntax, to reduce the "line noise"
  | effect. Speaking of line noise, how about saying that by convention, the '~'
  | ("null") can be used as a key to denote comments:
  |
  |     map: %
  |         ~ : This map contains this and that...
  |         key : value
  |         ...
  |
  |     delivery:%: 10-JAN-2001
  |         #: date
  |         ~: This is a comment about the delivery date
  |
  | Comments are just another color! This solves all our problems with them
  | (being in the info model on the one hand, but being ignored by applications
  | on the other). I *love* the color idiom.

  I do as well.  It may seem like extra noise if
  you are just worried about classes... however,
  when you start to add other colors, it becomes
  a very powerful mechanism...

  Hmm.  Just thinking.  The pattern above could work
  for any indicator used as a key...

    delivery: !date #"Comment on delivery" 10-JAN-2001

  | Now, if there's a way to de-serialize the above into something which behaves
  | "like a string" in Perl/Python...

  For Python, I think it works well.  All maps with
  a recognized "class" attribute are converted to
  a class on load (if they are recognized) otherwise
  they are left as maps.

  This works well.

  Best,

  Clark

  P.S.  I think I'd like to use # for comment and ! for class,
        what do you think?
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-03 16:06

```
  On Sun, Jun 03, 2001 at 04:57:10PM +0200, Oren Ben-Kiki wrote:
  | > I guess we should add \n to the list of escape sequences?
  |
  | It belongs together with \r, \t and the rest of them. No helping it. (\e for
  | ESC would be nice - I often wished for it when writing VT100 escape codes

  ok.  we add \r and \e

  | > Hmm.  Assume a binary file, containing only
  | > "x\n" on a unix box.  When this gets moved
  | > to a dos box, it will be "x\r\n".  Hmm.
  |
  | The trap of "looking like text". Right.
  |
  | > I guess YAR would have to do something
  | > special here, like encode if new lines
  | > are \n or \r\n as well as the character
  | > encoding.  Icky.
  |
  | If you realize it is a binary file, the only "special thing" you need to do
  | is encode it as a base64 blob. It will then be written as "x\n" in DOS as
  | well. The problem is, how do you realize "x\n" is a *binary* file
  | containing, say, a short integer whose value unfortunately looks ASCII text
  | when serialized?
  |
  | I guess the short answer is "you don't". The longer one is "give YAR all
  | sort of options, flags etc. to allow it to cope" - e.g., a '-b <path>'
  | forcing the file <path> to be handled as binary, or '-x <bin-ext>' forcing
  | all files ending with <bin-ext> to be handled as binary, etc.
  |
  | Pretty it isn't, but that's what you get from trying to handle CP/M newlines
  | rules, UNIX file systems, and not having BeOS mime type associations with
  | files...

  Well, it's either this or give up on the new line
  equivalence.  I think the new line equivalence
  is more important... YAR is a much smaller use case.

  Clark
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-03 16:08

```
On Sun, Jun 03, 2001 at 09:44:47AM +0200, Oren Ben-Kiki wrote:
| Things about the latest one which we agreed to fix:
|
| - Indentation in quoted strings;
| - Separate text from binary in the information model;
| - Clearing up how the API handles "multi-map" files (with blank lines).
|
| Other things which I noticed:
|
| - The draft requires a blank line at the end of the document;
| - There are some formatting issues;

Could you fix the above and send me the draft?  I'm
totally swamped with my day job...

| - There's a special standing for ISO=8859-1 which I don't follow. Isn't
| everything in Unicode? What does it mean to write an ISO-8859-1 escape
| within a quoted string?

Ahh.  There are two escapes, \xXX and \uXXXX, where \x takes
a 8 bit value and \u takes a 16 bit value.  You need both
since \x20BABE Is a space followed by BABE, where \u20BABE
is an oriental character (20BA) followed by BE.

| And, there's the class issue...

A "clean" proposal would help.  I know you made
a few in the past, but perhaps if we put this
on the table we could come to a conclusion.

;) Clark
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-03 16:21

```
On Sun, Jun 03, 2001 at 09:12:11AM +0200, Oren Ben-Kiki wrote:
| It seems I'm making a distinction here that you aren't.
|
| Issue 1: How to convert YAML text to an in-memory object
| and vice-versa (YAML issue).
|
| Issue 2: How to convert YAR file content represented in a
| YAML scalar to a file on the disk and vice versa (YAR issue).

Nice separation.

| A "binary" file will be represented as a base64 blob in
| the YAR file. You don't have to go that far to handle an
| occasional \r\n or \n in a file; you can just use a quoted
| string and escape them properly. That's still "mostly text".

I guess we should add \n to the list of escape sequences?


| > This gives the expected behavior when moving files
| > between platforms.  By the way, with the current
| > treatment, any YAML saved on a Windows box will
| > have \r\n line endings.  And likewise, on a
| > Unix box it will have \n line endings.  Since the
| > parser doesn't care, one can move the files back
| > and forth without affecting the canonical form.
|
| That's neat.

It's just a strawman.

| > This may not give exactly the same behavior
| > as "tar", since the exact line endings won't
| > be preserved when switching platforms... but this
| > is always a sore spot anyway!  As for "diff",
| > there are (or should be) flags to not report
| > line ending differences.
|
| So, YAR is more like "shar" then "tar".
| That makes a lot of sense.

Hmm.  Assume a binary file, containing only
"x\n" on a unix box.  When this gets moved
to a dos box, it will be "x\r\n".  Hmm.
I guess YAR would have to do something
special here, like encode if new lines
are \n or \r\n as well as the character
encoding.  Icky.

Clark

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

Re: [Yaml-core] yar

From: Brian Ingerson <briani@Ac...> - 2001-06-03 20:44

```
"Clark C . Evans" wrote:
>
> On Sun, Jun 03, 2001 at 05:15:56PM +0200, Oren Ben-Kiki wrote:
> | I had this notion about the syntactical form of class-as-color (or any
> | color). What if one allows the first key pair to appear immediately after
> | the '%':
> |
> |     delivery: % : 10-JAN-2001
> |         # : date
> |
> | OK, the class has to appear on a separate line, but that's not *too* bad,
> | textually. Especially after removing some whitespace:
> |
> |     delivery:%: 10-JAN-2001
> |         #: date
>
> If we want to "abbreviate", why not just allow
> the single line form...
>
>   delivery: !date 10-JAN-2001
```

```
>
> to be an abbreviation for ...
>
>   delivery: %
>      !  : date
>      "" : 10-JAN-2001
>

I wrote my last email suggesting "inline classes" without even seeing
this. We must be on a similar wave length. :)

However, be aware that we are suggesting two different things.

You are suggessting an abbreviation of coloring.

I am suggesting a "lightweight" semantic for Inline classes. One that
would not round trip between Perl and Java, but would round trip between
Perl and Perl. It's a disposable class I guess, at least from the YAML
side. Something like this will be vital to allow scripting languages (or
at least Perl) to use objects in their normal, lightweight way.

FYI, In Perl an object is just any old data structure with a name
attached by the bless() function. It can access it's data in through
predefined methods or not. It actually turns out to be a very compelling
OO model. But that's a different discussion.

Now Perl can also benefit from a heavyweight colored YAML object. For
instance, to redefine the emitter sort order, or to ensure that the
emmitter uses the correct encoding. Or for round tripping data to Java.

But the bottom line is that if you want to keep Perl interested (and me
too) you *need* a lightweight semantic option.

BTW, If the Perl implementation is done right, it will make YAML so much
more appealling tham XML.

Cheers, Brian

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

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-04 01:02

```
On Sun, Jun 03, 2001 at 11:26:29AM -0700, Brian Ingerson wrote:
| >
| >   delivery: #date 10-JAN-2001
| >
| > to be an abbreviation for ...
| >
| >   delivery: %
| >      # : date
| >      = : 10-JAN-2001

(the above fixed so that # means class, and = means
default value.)

| However, be aware that we are suggesting two different things.
| You are suggessting an abbreviation of coloring.

Right.

| I am suggesting a "lightweight" semantic for Inline classes. One that
| would not round trip between Perl and Java, but would round trip between
| Perl and Perl. It's a disposable class I guess, at least from the YAML
| side. Something like this will be vital to allow scripting languages (or
| at least Perl) to use objects in their normal, lightweight way.

Ok.  Why wouldn't the abbreviation work for round tripping
between Perl and Perl?  Would you ever write out a "map"
that has only a class (#) and a default value (=) ?
```

```
| BTW, If the Perl implementation is done right, it will make YAML
| so much more appealling tham XML.

I hope so!

;) Clark
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-04 03:04

```
On Sun, Jun 03, 2001 at 07:18:32PM -0700, Brian Ingerson wrote:
| > On Sun, Jun 03, 2001 at 11:26:29AM -0700, Brian Ingerson wrote:
| > | >
| > | >   delivery: #date 10-JAN-2001
| > | >
| > | > to be an abbreviation for ...
| > | >
| > | >   delivery: %
| > | >      # : date
| > | >      = : 10-JAN-2001
| >
| > (the above fixed so that # means class, and = means
| > default value.)
| >
| > | However, be aware that we are suggesting two different things.
| > | You are suggessting an abbreviation of coloring.
| >
| > Right.
| >
| > | I am suggesting a "lightweight" semantic for Inline classes. One that
| > | would not round trip between Perl and Java, but would round trip between
| > | Perl and Perl. It's a disposable class I guess, at least from the YAML
| > | side. Something like this will be vital to allow scripting languages (or
| > | at least Perl) to use objects in their normal, lightweight way.
| >
| > Ok.  Why wouldn't the abbreviation work for round tripping
| > between Perl and Perl?  Would you ever write out a "map"
| > that has only a class (#) and a default value (=) ?
|
| Yes. Perl to Perl *would* round trip but it wouldn't be required to.
| Basically YAML allows the abbreviation but doesn't require it to round
| trip. It is used for implementation specific round tripping. Perl and
| Python *could* use it if they agreed upon an implementation.

Ok.  So the abbreviation would be acceptable, rather
than having a separate mechanism then? In other words,
the abbreviated form and the non-abbreviated form would
appear *identical* as far as the sequential "parser"
API is concerned.  Is this ok?

It would then be up to the "Perl Binding" to specify
that both of the two forms above turn into a
delivery "scalar" having value 10-JAN-2001
blessed as a "date" class.

| The color scheme, on the other hand, represents the YAML object model,
| which every implementation is required to parse and preserve.

I'd like both forms to be YAML object which every implementation
must parse and preserve, one is just "syntax sugar" and is
not exposed in the API.

Yes?

Clark
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-04 10:02

```
Brian Ingerson [mailto:briani@...] wrote:
> Now Perl can also benefit from a heavyweight colored YAML object. For
> instance, to redefine the emitter sort order, or to ensure that the
> emmitter uses the correct encoding. Or for round tripping
> data to Java.

Cool.

> But the bottom line is that if you want to keep Perl
> interested (and me
> too) you *need* a lightweight semantic option.

Granted. However, I don't see why Clark's "abbreviated" syntax can't serve
as a "lightweight semantic option".

I thought the idea is that:

    price: !real 10.5

And:

    price: %
        = : 10.5
        ! : real

Will both be de-serialized, in Perl, into a simple number "10.5", in Java
into a simple Float 10.5, etc. A reasonable emitter will use the first,
shorter form. That seems like a "lightweight semantic option" to me... What
is the additional requirement you have for such an option that the above
mechanism doesn't provide?

(I'm using '!' for class so we can use '#' for comments - I think that's a
more intuitive use).

> BTW, If the Perl implementation is done right, it will make
> YAML so much
> more appealling tham XML.

Amen to that :-)

Have fun,

    Oren Ben-Kiki
```

Re: [Yaml-core] yar

From: Brian Ingerson <briani@Ac...> - 2001-06-05 05:48

```
Hi Oren,

After talking to Clark I agree that the abbreviation is now just an
abbreviation of color. Substituability mandates my change of mind. I
think it's of value but read on...

Oren Ben-Kiki wrote:
> > But the bottom line is that if you want to keep Perl
> > interested (and me
> > too) you *need* a lightweight semantic option.
>
> Granted. However, I don't see why Clark's "abbreviated" syntax can't serve
> as a "lightweight semantic option".
>
> I thought the idea is that:
>
>     price: !real 10.5
>
> And:
```

```
>
>     price: %
>         = : 10.5
>         ! : real
>
> Will both be de-serialized, in Perl, into a simple number "10.5", in Java
> into a simple Float 10.5, etc. A reasonable emitter will use the first,
> shorter form. That seems like a "lightweight semantic option" to me... What
> is the additional requirement you have for such an option that the above
> mechanism doesn't provide?

I think of 'real' as more of a special type. What about Foo?

If we have:

    key : !Foo a fooish string

That means I need to generate a Foo object which is a subclass of a YAML
object. All nodes will be objects which are subclasses of YAML. That's
because of this substituability thing. They pretty much have to be,
right? So my Perl might look like:

    $y = YAML->parse('doc1.yaml');
    print $y->getValue('key')->getValue;

prints:

    a fooish string

That's fine in Java, but it's way too ugly for Perl programmers to
accept. They'll want:

    print $y->{key};

Now this is just the simplest example. It gets really bad, really quick.

Even this isn't simpler:

    key : a fooish string

You would still need the exact same code as above, because of
substitutability it might turn into:

    key : @
        a fooish string
        a barrish string

Setting up Perl data for emission is just as problematic.

Now Perl has a wonderful "tie" facility that may allow me to expose the
simpler syntax. I'm not sure how well it will work out yet. But the
underlying implementation will be sloooow. C parser or not. In effect,
YAML no longer maps directly into scripting languages. It maps into OO
languages. A map is no longer a map. Its a YAML object. Same with
everything else.

So I'm willing to try this, but I'm skeptimistic about the usability
from Perl. This will never become a good alternative to Data::Denter.
(Long live Data::Denter :)

Perhaps a better alternative is to only provide substitutability within
classes. Therefore all classes will be first class YAML classes
(round-tripping, substitutable, etc), but regular data will never get
upgraded into a class. That way, maps are just maps like before. Hmm. I
really like that.

Maybe that's what you guys were already thinking? I'll have to go back
and check.

>
> (I'm using '!' for class so we can use '#' for comments - I think that's a
> more intuitive use).

That makes sense.


Cheers, Brian

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

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-05 07:20

```
Brian Ingerson [mailto:briani@...] wrote:
> I think of 'real' as more of a special type. What about Foo?
>
> If we have:
>
>     key : !Foo a fooish string
>
> That means I need to generate a Foo object which is a
> subclass of a YAML
> object.

If 'Foo' is a recognized class, you just build a Foo object. 'Foo' doesn't
have to derive from any YAML class.

If 'Foo' is an *unrecognized* class, then you build a YAML-object object.
The 'YAML-object' class (or whatever we'll call it) behaves like both a
string an a map. In an "ideal" language (Sather, anyone? :-) all you'd need
to do is write:

class YAML-OBJECT: public MAP, public STRING { some tricks to make it work
};

Again, there's no new class created; the above class is used only once,
throughout the system.

> All nodes will be objects which are subclasses of YAML.
> That's because of this substitutability thing.

Nope. Let's enumerate:

map: %
    key: value

Is de-serialized to a normal, built-in hash.

list: @
   value
   value

Is de-serialized to a normal, built-in list.

key : value

Is de-serialized into a normal string.

map: %
    = : value

And:

key : !class value

Are de-serialized into a YAML-object. That is, *Only* if you are actually
*using* substitutability, you pay for it (The good old C++ notion of "don't
use, don't pay"). That's why I don't like Clark's idea of a list having a
default value of the first element; it forces one to always pay this price,
when for most lists it isn't required. I also don't think it is useful to
evolve a scalar into a list; that's not part of the "color" idiom, anyway.
Clark?

> If we have:
>    key : !Foo a fooish string
>
> ... my Perl might look like:
>
>     $y = YAML->parse('doc1.yaml');
>     print $y->getValue('key')->getValue;
```

```
Ideally, if a proper YAML-object class can be written in Perl, it would look
like;

    # Access to "default" value of 'key' - 'fooish string'
    print $y->{key};
    # Access to value of 'key' as a map, to the class
    # member - 'Foo';
    print $y->{key}->{'!'};
    # Access to value of key as a map, to the value
    # member - 'fooish string'.
    print $y->{key}->{'='};

The question is: is this possible in Perl? I suspect it can...

> Even this isn't simpler:
>
>     key : a fooish string
>
> You would still need the exact same code as above
[i.e., the same code as accessing a YAML-object]

On that we agree; the question is, how simple can we make this code be, and
how efficient *in the normal case*. I'd rule out any implementation where
the above isn't de-serialized into a built-in Perl string.

> Now Perl has a wonderful "tie" facility that may allow me to
> expose the
> simpler syntax. I'm not sure how well it will work out yet. But the
> underlying implementation will be sloooow.

I've no problem with an implementation where an *unknown*-class-annotated
value is working slower. Makes sense?

> So I'm willing to try this, but I'm skeptimistic about the usability
> from Perl. This will never become a good alternative to Data::Denter.
> (Long live Data::Denter :)

I think that YAML *must* be usable from Perl. Let's make it so!

> Perhaps a better alternative is to only provide
> substitutability within
> classes. Therefore all classes will be first class YAML classes
> (round-tripping, substitutable, etc), but regular data will never get
> upgraded into a class. That way, maps are just maps like
> before. Hmm. I really like that.

I'm not certain exactly what you mean...

> Maybe that's what you guys were already thinking? I'll have to go back
> and check.

It does sound a bit like what I described above...

The only problem with the above scheme is that given a normal Perl map,
suppose I "innocently" add the key '=' to it. Ideally, it should polymorph
into a YAML-object. Deleting that key should cause it to polymorph back into
a normal map. It seems to me that either of these operations is momentous
enough to warrant an explicit call, such as:

 $yaml_object = make_yaml_object_from_normal_map($normal_map, $value);
 ($value, $normal_map) = make_normal_map_from_yaml_object($yaml_object);

Acceptable?

> > (I'm using '!' for class so we can use '#' for comments - I
> think that's a
> > more intuitive use).
>
> That makes sense.

OK, I'm going to do an update to the draft (nothing about the
substitutability yet, just the syntactical issues we agreed on) - I'll throw
that in as well.

Have fun,

    Oren Ben-Kiki
```

Re: [Yaml-core] yar

From: Brian Ingerson <briani@Ac...> - 2001-06-05 08:07

```
Oren Ben-Kiki wrote:
>
> Brian Ingerson [mailto:briani@...] wrote:
> > I think of 'real' as more of a special type. What about Foo?
> >
> > If we have:
> >
> >     key : !Foo a fooish string
> >
> > That means I need to generate a Foo object which is a
> > subclass of a YAML
> > object.
>
> If 'Foo' is a recognized class, you just build a Foo object. 'Foo' doesn't
> have to derive from any YAML class.

But it probably should. That's where it gets it's default behavior. Like
sort_order, etc.

> If 'Foo' is an *unrecognized* class, then you build a YAML-object object.
> The 'YAML-object' class (or whatever we'll call it) behaves like both a
> string an a map. In an "ideal" language (Sather, anyone? :-) all you'd need
> to do is write:
>
> class YAML-OBJECT: public MAP, public STRING { some tricks to make it work
> };
>
> Again, there's no new class created; the above class is used only once,
> throughout the system.

OK. I think I follow.

>
> > All nodes will be objects which are subclasses of YAML.
> > That's because of this substitutability thing.
>
> Nope. Let's enumerate:
>
> map: %
>     key: value
>
> Is de-serialized to a normal, built-in hash.
>
> list: @
>    value
>    value
>
> Is de-serialized to a normal, built-in list.
>
> key : value
>
> Is de-serialized into a normal string.
>
> map: %
>     = : value
>
> And:
>
> key : !class value
>
> Are de-serialized into a YAML-object. That is, *Only* if you are actually
> *using* substitutability, you pay for it (The good old C++ notion of "don't
> use, don't pay"). That's why I don't like Clark's idea of a list having a
> default value of the first element; it forces one to always pay this price,
> when for most lists it isn't required. I also don't think it is useful to
> evolve a scalar into a list; that's not part of the "color" idiom, anyway.
> Clark?

Well, I *like* all of this. It's certainly more appealing than what I
had feared. If non-classed structures remain native, we're cooking with
gas!
```

```
Clark's idea of having a scalar change to a list, requires that it
actually be neither. It can only be an object.

Also, it doesn't seem to make sense that a !Foo could morph into a !Bar
does it. Maybe if !Bar were a subclass...??? Do we support subclasses?
[...]
> > Perhaps a better alternative is to only provide
> > substitutability within
> > classes. Therefore all classes will be first class YAML classes
> > (round-tripping, substitutable, etc), but regular data will never get
> > upgraded into a class. That way, maps are just maps like
> > before. Hmm. I really like that.
>
> I'm not certain exactly what you mean...
>
> > Maybe that's what you guys were already thinking? I'll have to go back
> > and check.
>
> It does sound a bit like what I described above...

I think for once, Oren and I are on the same side of the fence, and
Clark might be on the other.

Clark?

Cheers, Brian

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

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-05 08:23

```
Brian Ingerson [mailto:briani@...] wrote:
> > If 'Foo' is a recognized class, you just build a Foo
> > object. 'Foo' doesn't
> > have to derive from any YAML class.
>
> But it probably should. That's where it gets it's default
> behavior. Like
> sort_order, etc.

We must ensure that (de)serialization works for arbitrary objects. This
isn't difficult for languages such as Perl, Python and even Java. Reflection
capabilities which are built-in to the language allows you to do a
reasonable job, adequate for most objects.

Sure, we'll end up defining an interface which, if an object supports,
allows it to control the (de)serialization - what fields are printed first,
say, or not printing "derived" fields and re-computing them when the object
is read, etc. But that's luxury. The principle is "anything goes". The Java
serialization model is a good example of how such things can be done.

It may be worth it to consider subverting the Java mechanism for our
purposes somehow. Imagine every Java object in the world being YAML-aware
"out of the box". I know someone did something similar for XML - maybe it
can be used as a basis. The only problem is that it will litter the files
with too many Java-specific class names, and too many anchors... Hmmm...
This deserves some serious thought for which I don't really have the time.

> Clark's idea of having a scalar change to a list, requires that it
> actually be neither. It can only be an object.

Yup. That's why I dislike it.

> I think for once, Oren and I are on the same side of the fence, and
> Clark might be on the other.

It seems that way. Well, the (dis)advantage of a triumvirate is that two
people can always gang up on the third :-)
```

```
Have fun,

    Oren Ben-Kiki
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-05 12:15

```
On Tue, Jun 05, 2001 at 01:07:16AM -0700, Brian Ingerson wrote:
| > Are de-serialized into a YAML-object. That is, *Only* if you
| > are actually *using* substitutability, you pay for it (The
| > good old C++ notion of "don't use, don't pay").

I was only considering the iterator (parser)
interface having this functionality built-in,
since the additional overhead is an additional
(and rather simple) function that the user
of the library can just ignore if they so choose.

For language bindings, I was thinking that the
asScalar function would be more or less an *external*,
or "friend" function, not a method. I was using the
method syntax for readability, I had not intended
that it actually be a method....

If, for some cases, it can be implemented as
a method, well... more power to that language.
But I don't think this is at all a requirement,
an external function should be just fine, as ugly
as it may be.

| > That's why I don't like Clark's idea of a list having a
| > default value of the first element; it forces one to always
| > pay this price, when for most lists it isn't required. I also
| > don't think it is useful to evolve a scalar into a list;
| > that's not part of the "color" idiom, anyway.

I think it may be useful for times when something originally
thought to be single value (such as an address) turns out
to be multi-valued.  If you only allow "extensions" using
the map construct, this forces a particular way to extend
the scalar (use a map, and then a list) instead of just
converting the scalar into a list.  I've had this case
pop up before...

| > Clark?

Well, if it seems problematic... I'll compromise here. ;)

| Well, I *like* all of this. It's certainly more appealing than what I
| had feared. If non-classed structures remain native, we're cooking with
| gas!

I hope so.  Charcol may taste better, but it
is very messy.  And who likes to chop their own wood?
For recreation, yes, but not for daily work!

| Clark's idea of having a scalar change to a list, requires that it
| actually be neither. It can only be an object.

Brian, thank you for working the proposal out.... I
was not actually intending this.  Although I was
thinking of an exteranal "asScalar()" function that
returns a scalar when handed a list.  Is this possible?

| Also, it doesn't seem to make sense that a !Foo could morph
| into a !Bar does it. Maybe if !Bar were a subclass...??? Do
| we support subclasses?

Hmm. Suppose you were expecting...

  tag:
    a: "hello"
```

```
and someone gave you,

  tag:
    a: "hello"
    b: "world"

Then the second case is substutable for tag.  Hmm.
How can this be codified?


| > > Perhaps a better alternative is to only provide
| > > substitutability within classes. Therefore all
| > > classes will be first class YAML classes
| > > (round-tripping, substitutable, etc), but regular
| > > data will never get upgraded into a class. That way,
| > > maps are just maps like before. Hmm. I really like that.
| >
| > I'm not certain exactly what you mean...

I was thinking that the asScalar function is
not a method... and thus completely separate
from both the native types and classes.  I hadn't
actually put serious mental energy into thinking
how it would work for a given class system.  If
you *have* a YAML class... then I can see where
asScalar() could be a nice method.

| I think for once, Oren and I are on the same side of the
| fence, and Clark might be on the other.

You will find my perspective leans towards
the iterator (parser) interface... which is
quite a bit different from the native,
internal representation.

I think we are all on the same side of the fence....

Best,

Clark
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2001-06-05 13:21

```
Clark C . Evans [mailto:cce@...] wrote:
> | > That's why I don't like Clark's idea of a list having a
> | > default value of the first element; it forces one to always
> | > pay this price, when for most lists it isn't required. I also
> | > don't think it is useful to evolve a scalar into a list;
> | > that's not part of the "color" idiom, anyway.
>
> I think it may be useful for times when something originally
> thought to be single value (such as an address) turns out
> to be multi-valued.  If you only allow "extensions" using
> the map construct, this forces a particular way to extend
> the scalar (use a map, and then a list) instead of just
> converting the scalar into a list.  I've had this case
> pop up before...

An example would be nice...

> | > Clark?
>
> Well, if it seems problematic... I'll compromise here. ;)

So can we say its settled, subject to Brian confirming it is implementable?
Brian?

> | Also, it doesn't seem to make sense that a !Foo could morph
> | into a !Bar does it. Maybe if !Bar were a subclass...??? Do
> | we support subclasses?
>
> Hmm. Suppose you were expecting...
>
>   tag:
>     a: "hello"
>
> and someone gave you,
>
>   tag:
>     a: "hello"
>     b: "world"
>
> Then the second case is substutable for tag.  Hmm.
> How can this be codified?

Well, as long as you only add data members you are basically OK.

I wouldn't worry too much about it:

- If both classes aren't recognized, then as long as you just add members
you are OK;
- If both classes are recognized, then the language itself provides you with
compatibility. Presumably, the implementation of 'Foo' and 'Bar' will be
compatible if they are supposed to be. In C++, 'Bar' will derive from 'Foo'
and all will be well; in Perl, 'Bar' just has to provide all of 'Foo's
methods and members, etc.

So far, so good. The problems are:

- If 'Foo' is recognized and 'Bar' isn't, a twisted and heavyweight
mechanism may be able to help you somewhat;
- If 'Bar' is recognized and 'Foo' isn't, no force in the world can help
you.

I say the 80/20 solution here is just ignore these last two cases.

Have fun,

    Oren Ben-Kiki
```

Re: [Yaml-core] yar

From: Clark C . Evans <cce@cl...> - 2001-06-05 13:45

```
On Tue, Jun 05, 2001 at 03:21:51PM +0200, Oren Ben-Kiki wrote:
| > I think it may be useful for times when something originally
| > thought to be single value (such as an address) turns out
| > to be multi-valued.
|
| An example would be nice...

Examples:

  a) A phone number
  b) A contact person
  c) An address

In many of these cases, a business analyst will often
consider only one number, person, or address to be
what is needed.  But later on, it is discovered that
there are cases where more than one of the above
would be useful.  The contact person being one example
that has bit me before in a customer support database
that I was working on.   If you only allow a scalar to
become a map, then this will force an additional
(and unnecessary) intermediate map, as well as a
duplication of the first value...

  contact: Jamie Hedward

Becomes...
```

```
  contact: %
      =: Jamie Hedward
      list: @
         Jamie Hedward
         Jenny Millsap

instead of...

  contact: @
     Jamie Hedward
     Jenny Millsap


| > Well, if it seems problematic... I'll compromise here. ;)
|
| So can we say its settled, subject to Brian confirming
| it is implementable?

I'm willing to compromise here since this isn't a
"common" thing to do... extending via map is far
more common (as maps are more common than lists).

Did you see my post on "asScalar" being an *external*
function and not a member function?  In this case, it
doesn't matter if the value is a map, scalar, or value.
Also bindings my differ from language to language.  As
long as the iterator interface supports the "asScalar",
I really don't care if the "native representation"
supports this construct (although it would be nice).

| > Hmm. Suppose you were expecting...
| >
| >   tag:
| >     a: "hello"
| >
| > and someone gave you,
| >
| >   tag:
| >     a: "hello"
| >     b: "world"
|
| Well, as long as you only add data members you are basically OK.

I like your breakdown, nice.

| I say the 80/20 solution here is just ignore these last two cases.

Good deal.

Clark


P.S.  Thank you for taking on the spec updates!
```

RE: [Yaml-core] yar

From: Oren Ben-Kiki <orenbk@ri...> - 2002-01-29 07:24

```
Brian Ingerson wrote:
> I've been thinking about a good format for 'yar', the YAML ARchive
> format that will be a readable and editable substitute for tar or zip.

Our trustworthy use case. It is a good app to demonstrate YAML, and is
useful besides. Thanks!

> My basic format idea is to have the first document in the yar stream be
> a directory containing all the necessary file system info. The directory
> entries will be an index pointing to other documents in the stream. Each
> following document will be the contents of a file. Plain text files can
> be stored as is. Binary docs can be encoded in base64 or something else.

You seem to assume that the YAR writer has random access to the data...
which seems reasonable. By placing the directory at the start you force the
```

writer to do a two-pass (at least) on the data, at the very least building a
full TOC in memory or re-accessing it when emitting the YAR file.

It does allow a reader application to quickly show you a TOC and given a
byte/character "start at offset" field, may allow very quick random access
for extracting one particular file.

However there may be a problem if the file system changes between the two
passes of the writer program (if a file size changes, if a file is
added/removed/renamed, etc.). You could work around it by emitting the TOC
to one file and the content to another in a single pass, then concatenating
both files into a single stream... But this prevents one from writing "yar
--create | whatever". Of course you could just treat changes between passes
as a warning/error... Thoughts?

What WinZip does is place the directory at the end of the stream. This
allows the writer to do a single-pass on the data, so there are no
consistency problems. But it prevents one from writing "whatever | yar
--extract". It still allows quick access to files if there's random access
to the YAR file, however (finding a trailing TOC is harder but still
possible).

The only true single-pass solution would be for YAR to directly reflect the
directory structure of the data. Something like:

```
!yar/dir /etc :
 perm: rw-r--r--
 files:
  !yar/file passwd :
   ...
  !yar/dir rc.d :
   ...
```

This means that there would be no quick way to extract a TOC out of a file,
however. Sigh. A tradeoff... what's more important, "yar --create | yar
--extract" or a very fast GUI-based YAR selective extraction utility? Is
"yar --create | whatever" less important than "whatever | yar --extract"?

BTW, you could make a YAR stream always consist of two consequtive
documents. The first would be a tree as above. The second (optional?) would
be a TOC. This would allow both single-pass read/write and the ability to do
TOC operations. The cost is replication of data, and added complexity. But
it may be worth it...

> [indenting top-level leaf values]
> This is a powerful notion. If a document conatins '^---', we no longer
> need to switch to a new separator. We can simply indent the top level
> scalar! This is good because the stream might already be started by the
> time we realize that there is a conflict.

Hmmm. Does this mean we can make '---' be THE separator, period, instead of
auto-detecting one? Clark?

> I'd like to know if anyone has ideas on yar in general.

I think we should take a look at the design of shar, tar and cpio before we
commit to the details... I remember reading a good document about the design
of cpio but I've no idea where to find it (it was years and years ago). A
quick search didn't show up anything useful. Also I know many people have
notions about what tar and cpio do wrong... We should take the time to
become familiar with this and "do better", just like we did with YAML vs.
XML :-)

I'll do some more digging for for relevant stuff the minute I finish (sigh)
updating the spec to cover option (E) and fix the productions bugs (yes,
there are some. Sorry about that).

Off the top of my head I think we should have (optional?) CRCs of some sort;
allow specifying a mime type per file (optional, I guess); in fact we may
want to use an "ignore unknown properties" policy to allow round-tripping
features within a platform with interoperability outside it (I think this is
a basic YAML idiom). A nice test case for this would be handling of Mac
resources so they round-trip on Macs and would look like a pair of files
elsewhere. I'd also consider round-tripping BeOS files as a gedanken use
case (BeOS being dead in the water means it probably won't ever become a
real one).

The interaction with compression is also interesting. Applying compression
to the YAR file as a whole removes many readability advantages of the
format... It seems if we compress each file content we may get most of the
size advantage while still having a readable file... except when there are
many small files... and when using the "tree" approach the indentation would

```
be a horrible waste... Hmmm.

We could just allow for it *and* allow an optional "transfer encoding"
property for each file anyway, so it would be the YAR writer choice which to
use. It seems best...

> Perhaps this
> discussion should be taken to yaml-dev or yaml-projects. Have thes
> (any) new mailing lists been started yet?

I'm not aware of any, and I think that until YAML is in a release candidate
state we can discuss such things here...

Have fun,

    Oren Ben-Kiki
```

Next Messages