lengthKind="pattern" Another delimiter expression or a complex character class (Answered)

The current implementation of `lengthKind="pattern"` assumes the `lengthPattern` expression stores a value to be match. That is to say if one wanted a hexadecimal string, one would set `lengthPattern="[0-9A-Fa-f]+"` in the current implementation. This would still require a separator attribute though because it doesn't code for the delimiter that would normally separate fields.

However, the AO and AV tests from Tresys assume the opposite: that the lengthPattern is another form of separator/delimiter and doesn't say anything about the data format for the given field.

> MikeB said: The correct definition is 'value to be matched', not a regex way to express delimiters.

In the current implementation, the separator might also become part of the pattern since it's possible to define a positive look-ahead which specifies the end of the field. For example, if the separator is a comma (`,`) and the pattern is `"[A-Z][a-z]*(?=,|$)"`, (meaning the next field must consist of a capital/majuscule letter followed by zero or more lower-case/minuscule letters which then terminates at the first comma or end of string.

The `testLengthKindPattern` test case in `sub-projects/core-srcTest/daffodil/api/TestAPI1.scala` has another example of how the current implementation might be used:

```
lengthPattern=".*?[^\\](?=,|$)"
```

The above expression says match any character, zero-or-more though not more than necessary, followed by any character that isn't a backslash (/). It then asserts that whatever follows this (the first instance of this possible match thanks to the "though not more than necessary, non-greedy" clause) is either a comma or an end of string.

If the code were changed to mean that lengthPattern is just another word for a regex separator, this would have to be changed to:

```
lengthPattern="(?<!\\),"
```

Which uses a negative look-behind to ensure that the given comma delimiter is not immediately preceded by a backslash. The looks behinds though have to be fixed length, unlike the look-aheads due to limitations in more regex parsers, so for instance (?<=a{1,3}) is not a valid positive look-behind.

So the opened question is, which is better: should lenghtPattern subsume separator or should it specify the pattern the field may match?

~~Note: changing the code to support pattern is delimiter would not be that much work so time to implement should not be a consideration.~~

~~I have made the necessary local changes to reverse the logic to make it a search by regex delimiter and still find the AO000, AO001, AO002, AO003, AO004, AV000, AV001 and AV002 tests are still failing due to an inability of the parser to find the initiator property.~~

And I reverted back to the original implementation and test thanks to Mike's comment below; we will treat lengthKind="pattern" as a regex match against the field. What has yet to be decided is how to handle field delimiters and separators.

> MikeB said: Best to go to the current draft of the DFDL spec. The AA-BG tests were part of the original daffodil code base, and that was developed based on a very early draft of the DFDL standard, which has since changed a great deal as it has converged toward an agreed standard. In particular, way back somebody suggested "why not let delimiters be regular expressions?" For a while this was entertained, and that's when the original daffodil code base was created along with tests AA-BG. But this was later viewed as too much generality (hence complexity, especially when debugging "why doesn't my DFDL schema work???"), for not enough benefit.
>
> The lengthKind="pattern" was added to handle the cases which truly need a complex match.